FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Teacher Assignment Problem at DEI

Pedro Miguel Vieira da Câmara

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Daniel Castro Silva

August 12, 2018

Teacher Assignment Problem at DEI

Pedro Miguel Vieira da Câmara

Mestrado Integrado em Engenharia Informática e Computação

August 12, 2018

Abstract

Every year the Department of Informatics Engineering (DEI) of the Faculty of Engineering of the University of Porto (FEUP) has to assign each teacher to the classes which are to be taught. This is currently done manually, mostly by just slightly changing the previous year assignments. The current process does not take into account the teachers' preferences which in turn lowers their satisfaction. This problem also exists in all the other departments at the faculty and in many other universities. This common problem is known as the Teacher Assignment Problem.

The Teacher Assignment Problem is part of the Course Scheduling Problem which can be decomposed into five different ones: Course Timetabling, Class-Teacher Timetabling, Student Scheduling, Teacher Assignment and Classroom Assignment. However, the timetabling process is somewhat unique to each institution and the different sub-problems can be solved in different orders from institution to institution. In the case of DEI, we can consider three different stages, in the following order: Teacher Assignment, Course Scheduling and Student Scheduling. Besides these, a fourth stage can also be considered consisting of an Exam Scheduling Problem.

The objective of the work developed herein is the implementation of a system which allows teachers to state their preferences and which is able to automatically assign teachers to course sections. The system supports multiple types of preferences and is modular so that more preferences can be easily added in the future. The system also allows the provided solution to be manually refined afterwards if required. The different types of preferences and constraints make this teacher assignment problem more complex and difficult to solve than most in the literature.

A good system allows teachers to get classes and workloads that better fit their preferences. This can in turn help improve the quality of the lessons themselves since the teachers will be more interested in the courses they are responsible for.

The developed system uses constraint programming and takes into account the teachers' preferences in relation to courses, other teachers and workload across both semesters of an academic year. The performance experiments show positive results, with some metrics showing better performance than the manual assignments.

Keywords: Timetabling, Scheduling, Teacher Assignment Problem, Teacher-Class Assignment, Operational Research, Constraint Programming

Resumo

Todos os anos, o Departamento de Engenharia Informática (DEI) da Faculdade de Engenharia da Universidade do Porto (FEUP) tem de atribuir cada professor às aulas a lecionar. Atualmente isto é feito de forma manual, principalmente através de alterações feitas às atribuições do ano anterior. O processo atual não tem em consideração as preferências dos professores, o que pode, por sua vez, diminuir a sua satisfação. Este problema também existe em todos os outros departamentos da faculdade e em muitas outras universidades. Este comum problema é conhecido como o Problema da Distribuição de Serviço Docente.

O Problema da Distribuição de Serviço Docente faz parte do Problema de Agendamento de Unidades Curriculares, que pode ser decomposto em cinco problemas diferentes: Geração de Horários, Agendamento entre Professores e Turmas, Atribuição de Alunos a Turmas, Distribuição de Serviço Docente e Atribuição de Salas. No entanto, o processo de calendarização é algo único para cada instituição e os diferentes subproblemas podem ser resolvidos em diferentes ordens de instituição para instituição. No caso do DEI, podemos considerar três etapas distintas, na seguinte ordem: Distribuição de Serviço Docente, Geração de Horários e Atribuição de Alunos a Turmas. Além destas, uma quarta etapa pode ser também considerada, que consiste no problema de Agendamento de Exames.

O objetivo do trabalho aqui desenvolvido é a implementação de um sistema que permita aos professores definir as suas preferências e que seja capaz de atribuir automaticamente os professores a unidades curriculares e respetivos tipos de aulas. O sistema suporta vários tipos de preferências e é modular para que mais preferências possam ser facilmente adicionadas no futuro. O sistema também permite que a solução fornecida seja posteriormente editada manualmente, caso seja necessário. Os diferentes tipos de preferências e restrições tornam este problema de distribuição de serviço mais complexo e difícil de resolver do que a maioria na literatura.

Um bom sistema permite que os professores recebam unidades curriculares e cargas de trabalho que melhor se adequem às suas preferências. Isto, por sua vez, pode ajudar a melhorar a qualidade das aulas, já que os professores estarão mais interessados nas aulas pelas quais são responsáveis.

O sistema desenvolvido recorre a programação por restrições e tem em conta as preferências dos professores em relação a unidades curriculares, a outros professores e à carga de trabalho em ambos os semestres de um ano letivo. As experiências realizadas mostram resultados positivos, com algumas métricas a mostrar melhor desempenho que as atribuições manuais.

Acknowledgements

I would like to start by thanking my supervisor, Daniel Silva, for helping me with this dissertation and for always being available for answering questions and giving advice.

I would also like to thank my family for supporting me during these five years and for their patience, specially during these last few months in which I have been less present.

Finally, I would like to thank my friends for their help, as well as my beloved girlfriend for her encouragement and support through this journey.

Pedro Câmara

"If I have seen further it is by standing on the shoulders of giants"

Isaac Newton

Contents

List of Figures

List of Tables

Abbreviations

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CP	Constraint Programming
DEI	*Departamento de Engenharia Informática* (Department of Informatics Engineering)
FEUP	*Faculdade de Engenharia da Universidade do Porto* (Faculty of Engineering of the University of Porto)
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IdP	Identity Provider
IP	Integer Programming
JAR	Java Archive
JSON	JavaScript Object Notation
JSONB	JavaScript Object Notation Blob
KBS	Knowledge-Based System
LP	Linear Programming
MILP	Mixed Integer Linear Programming
MVVM	Model-View-ViewModel
ORM	Object-Relational Mapping
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SIGARRA	*Sistema de Informação para a Gestão Agregada dos Recursos e dos Registos Académicos* (Information System for Aggregate Resource Management and Academic Records) (UP's information system)
SSH	Secure Shell
SPA	Single Page Application
SQL	Structured Query Language
SSO	Single Sign-On
TAP	Teacher Assignment Problem
TLS	Transport Layer Security
UI	User Interface
UP	*Universidade do Porto* (University of Porto)
XML	Extensible Markup Language
XSRF	Cross-Site Request Forgery

Chapter 1

Introduction

This chapter provides an overview of this dissertation and the work developed herein. The context and motivation for this project are described followed by the problems this dissertation addresses. In the end a short summary of the chapters below is presented.

1.1 Context/Background

The topic of this dissertation was proposed by FEUP's Department of Informatics Engineering (*Departamento de Engenharia Informática*), also known as DEI.

Every year, the department has the task of assigning each course and respective lessons, which can be of different types, to the necessary teachers. This is currently done by hand and the general process consists of taking the previous year assignments and changing them by some intuition as well as trial and error. The current process does not take into account the teachers' preferences which in turn lowers their satisfaction. This problem is common to the other departments and many other universities and is known in the literature as the Teacher Assignment Problem.

The Teacher Assignment Problem is part of the Course Scheduling Problem. Carter and Laporte [1998] have proposed a classification for this problem: (1) Course Timetabling: to assign courses or course sections to time periods; (2) Class-Teacher Timetabling: to schedule class-teacher meetings without creating conflicts - this problem arises mostly in high schools; (3) Student Scheduling: to assign students to course sections, after they have chosen their courses, while balancing section sizes and respecting room capacities; (4) Teacher Assignment: to assign teachers to courses maximising a preference function; and (5) Classroom Assignment: to assign events to rooms which satisfy size, location and facility preferences and restrictions.

Timetabling problems are actually pretty common. Not only do they exist in schools and in universities, but also in hospitals, public transportation and many other institutions. Multiple approaches have been used to solve problems of this kind, some of the most common being integer programming and artificial intelligence heuristics.

Furthermore, each of these problems, including the Teacher Assignment problem, although seemingly simple at first, are actually relatively complex, being NP-hard problems [Avella and

Vasil'Ev, 2005]. For this reason, usually each problem is solved separately from the other problems related to the university timetabling. Each problem requires a significant number of variables and restrictions, which can be either soft or hard constraints, and a complicated evaluation function. The combination of multiple problems greatly increases the complexity of the problem to solve and the time to compute a quality solution.

1.2 Motivation and Objectives

As mentioned previously, in DEI the assignment of teachers to classes is currently done manually. This complex task, carried out by a single person, is done by trial and error and takes several days or even weeks to finalise. It is intended, therefore, to automate this process and at the same time to make it more flexible and to try to better satisfy the teachers' preferences.

In DEI, first the teachers are assigned to courses, afterwards the timetables are created and then the students are assigned to classes according to their preferences. While the timetabling and the student-class assignments are currently solved with the help of existing algorithms, the teacher assignment problem is still solved manually. This dissertation aims to automate this stage of the process. This turns the whole course scheduling problem into something that is mainly solved automatically, except for some necessary manual adjustments. The exam timetabling, which is considered a different problem, and which is done only after everything is scheduled, is also currently done manually.

The main objective of this work is the development of a platform which allows teachers to input their preferences through a web app and the administration of the department to automatically generate the assignments between teachers and course sections. In addition, this solution can be manually edited if necessary. The system supports three preferences types, which can be in relation to courses, other teachers and workload distribution across both semesters. The system aims to be extendable so that new preferences can be added in the future should that be required. The number of preferences and constraints to be implemented as well as the dimensions of the problem instance make this problem more complex and challenging to solve than most in the literature.

Some of the variables which are taken into account include the scientific area of the teachers and of the courses, the previous year assignments, the rank of each teacher, how many hours they want to teach per semester, which classes they prefer, among others.

This system aims to make the teachers more satisfied in terms of the classes and workloads they have, while making the process fairer. Some improvement in the quality of the lessons may also occur, as it is expected that the teachers will be more interested in the courses they are assigned to.

Furthermore, this project intends to check the feasibility of an automated solver for data sets of this dimension as well as the amount and types of constraints which can be added without jeopardising the ability of solving the problem in a reasonable time frame.

1.3 Dissertation Structure

The structure of this dissertation is organised as follows. Chapter 2 describes the problem in more detail. This includes the problem definition, the variables, restrictions and evaluation problem. In chapter 3 a literature review is presented as well as a description of relevant related work. Chapter 4 presents the planning for the solving of the problem at hand. Chapter 5 explains what was developed during the dissertation, including the features of the project and how they were implemented. In chapter 6 the results of the implementation are presented and discussed. Finally, chapter 7 closes this report by drawing some conclusions and by talking about possible future work.

Chapter 2

Teaching Assignment Problem

In this chapter a detailed explanation of the problem associated with this dissertation is presented. The multiple constraints, variables and evaluation function, with its multiple components, are also explained.

2.1 Fundamental Definitions

Before explaining the problem in more detail, some of the fundamental definitions essential to its understanding are presented below.

- **Teacher:** A teacher is someone who gives lessons at the university. Each teacher has an associated rank which can be one of the following: professor (*catedrático*), associate (*associado*), assistant (*auxiliar*), guest professor (*catedrático convidado*), guest associate (*associado convidado*), guest assistant (*auxiliar convidado*) and teaching assistant (*assistente convidado*). Furthermore, in the case of DEI, each teacher has a scientific area, such as Software Engineering or Intelligent Systems. In addition to teaching on their correspondent area, each teacher might be able to teach other classes corresponding to different areas.
- **Programme:** A programme consists on a plan of study on a particular subject, such as Informatics Engineering or Electrical Engineering. When completed, a programme provides the student with a degree. There are multiple types of degrees: Bachelor's, Integrated Master's, Master's and Doctoral Degree.
- **Lesson:** A lesson consists of a single period of time in which a subject is taught. There are multiple types of lessons: Practical-Laboratorial (*Prático-Laboratorial*, PL), Lecture (*Teórica*, T), Practical-Lecture (*Teórico-Prática*, TP), Internship (*Estágio*, E), Tutorial Guidance (*Orientação Tutorial*, OT), Seminar (*Seminário*, S), Field Work (*Trabalho de Campo*, TC) and Other (*Outro*, O). Some types of lessons, such as lectures, might require the assigned teacher to have a doctoral degree.
- **Course:** A course (*unidade curricular*) consists of a set of lessons on a more specific subject related to the programme it is inserted in. Each course usually lasts for a semester and has

a certain number of hours which are taught every week. In some cases courses last for less than one semester and in other cases during a whole academic year. Many courses have a mix of lectures and practical lessons but some have only lessons of a single type. Each course is associated with one or more scientific areas.

- **Class:** In the context of the university timetabling, a class is related to a lesson or set of lessons from a single course which are taught every week to the same group of students during a semester. These are usually taught by the same teacher ou group of teachers.
- **Section:** In the context of this problem, a section will be considered as a block which has to be assigned to a single teacher. This is usually a class or the lectures of a course, but it can also be just a part of class in the cases where two teachers are responsible for a single class. For example, a section can be 2 hours of a practical class for a specific course.

2.2 Problem Explanation

As mentioned in the previous chapter, the Teacher Assignment Problem is part of the Course Scheduling Problem [Carter and Laporte, 1998]. All its sub-problems are considered NP-hard since all timetabling problems have been proven to have that type of complexity [Avella and Vasil'Ev, 2005].

The teaching assignment problem consists of assigning the teachers to courses which are to be taught in a given academic year. This problem can be split into the following stages [Reis, 2003]:

1. Collection of the information about the teachers' qualifications and their multiple preferences as well as information about the courses and their requirements.
2. The assignment itself. This should take into consideration the preferences and information described above as well as all the other constraints which will be described later in this chapter. This includes, for example, the fact that each course needs to have the desired number of sections assigned.
3. Optimising the result of the previous stage, manually or not, by taking into account teachers' complaints and suggestions.

The solution to the teaching assignment problem can be defined as: for each course define the course lecturer and for each section of each course select a teacher responsible for it, while satisfying a collection of constraints.

2.3 Hard Constraints

In the teaching assignment problem both soft and hard constraints are used in order to improve the final result. However while soft constraints are possible to be violated, hard constraints have to be always satisfied.

The hard constraints in this case are the following:

- All course sections must have exactly one teacher assigned. If there are not enough teachers, a guest teaching teacher can be chosen for that section.
- The teachers must be within their minimum and maximum teaching hours according to their contract.
- The number of courses per teacher must not exceed the maximum value. This number can be reduced due to workload reductions attributable to management roles.
- The unavailability of the teachers in a given semester must be taken into account. This can be, for example, due to sabbatical, parental or unpaid leaves, amongst others.

2.4 Soft Constraints

The soft constraints, while not essential, help to improve the quality of the solution and are part of the evaluation function. They are the following:

- The teachers' course and course type preferences should be taken into account. These preferences can be either positive, negative or neutral.
- The number of hours per week per teacher must be within a given threshold. This is to ensure the expected workload is followed but also to provide some flexibility.
- The number of distinct courses per teacher must be within a given threshold. This is to avoid teachers from having either too few or too many courses. Few courses implies the teacher will have many hours in one course which makes the schedules creation more difficult, as those classes can not be taught in parallel. On the other hand, a large amount of courses would make the teacher have to prepare many different lessons.
- The number of hours to be taught by a single teacher in a single course should be within a given threshold. Similarly to the previous point, this is to avoid difficulties with the schedules creation.
- The teachers' preference regarding the teaching load distribution across both semesters must be taken into account.
- The teachers' teaching load should be balanced across all teachers as most as possible.
- The teachers' preferences regarding other teachers must be taken into account. Similarly to the course preferences, these can be either positive, negative or neutral.
- The number of hours assigned to guest teachers should be minimised;

- Programme directors' as well as scientific area coordinators' preferences regarding who should lecture each course should be taken into account.
- Teachers' ranks should be taken into account, with higher-ranked teachers having a higher weight on their preferences.
- The number of changes from one year to another, in terms of the courses assigned to each teacher, should be minimised.
- Other types of preferences from the direction.

2.5 Objective/Evaluation Function

The objective function uses the soft constraints as optimisation criteria. Due to the fact that it is this function which guides the solver to a solution and is what differentiates one feasible solution from another in terms of which one is better, special care is required when creating it, specially with the weights assigned to each component of the function.

2.6 Mathematical Model

In this section, a typical model for solving this problem is shown. It is modelled with integer programming, where binary variables represent the assignment of a teacher to a course section. This model is based on the one by Domenech and Lusa [2016].

The following parameters are used:

- T = Number of teachers
- C = Number of courses
- B = Number of blocks
- h_b = Number of weekly hours associated with *block b*
- $minh_t$ = Minimum number of weekly hours *teacher t* should be assigned to
- $maxh_t$ = Maximum number of weekly hours *teacher t* should be assigned to
- $course_b$ = Course associated with each *block b*
- $course\_limit$ = Maximum number of courses per teacher
- $week\_limit$ = Maximum number of hours per week per teacher
- $q_{tb} = \begin{cases} 1 & \text{if } \textit{teacher t} \text{ is qualified to teach } \textit{block b} \\ 0 & \text{if } \textit{teacher t} \text{ is not qualified to teach } \textit{block b} \end{cases}$

- $a_{ts} = \begin{cases} 1 & \text{if } \textit{teacher } t \text{ is available in } \textit{semester } s \\ 0 & \text{if } \textit{teacher } t \text{ is not available in } \textit{semester } s \end{cases}$
- $s_b =$ semester (1 or 2) of a given *block b*

The defined decision variables are:

- $x_{tb} = \begin{cases} 1 & \text{if } \textit{teacher } t \text{ is assigned to } \textit{block } b \\ 0 & \text{if } \textit{teacher } t \text{ is not assigned to } \textit{block } b \end{cases}$
- $xc_{tc} = \begin{cases} 1 & \text{if } \textit{teacher } t \text{ is assigned to } \textit{course } c \\ 0 & \text{if } \textit{teacher } t \text{ is not assigned to } \textit{course } c \end{cases}$

The equations for the hard constraints are the following:

All course sections must have either 1 or 0 teachers assigned. A 0 means a guest teacher will teach that section:

$$\forall t \in [1,T], \forall b \in [1,B], x_{tb} \leq 1 \tag{2.1}$$

The teachers must be within their minimum and maximum teaching hours according to their contract:

$$\forall t \in [1,T], minh_t <= (\sum_{b=1}^{B} x_{tb} \times h_b) <= maxh_t \tag{2.2}$$

The number of courses per teacher must not exceed the maximum value:

$$\forall t \in [1,T], \sum_{c=1}^{C} xc_{tc} < course\_limit \tag{2.3}$$

The maximum number of hours per week per teacher must not be exceeded:

$$\forall s \in [1,2], \forall t \in [1,T], \sum_{n=1}^{B} (s_b = s) \times x_{tb} \times h_b \leq week\_limit \tag{2.4}$$

Teachers must not give a class they are not qualified for:

$$\forall t \in [1,T], \forall b \in [1,B], 1 - x_{tb} + q_{tb} \geq 1 \tag{2.5}$$

The unavailability of the teachers in a given semester must be taken into account:

$$\forall t \in [1,T], \forall b \in [1,B], a_{ts_b} >= x_{tb} \tag{2.6}$$

2.7 Summary and Conclusions

The Teacher Assignment Problem is part of the Course Scheduling Problem. Similarly to its other sub-problems, it has some complexity, specially with all the possible preferences which are to be

implemented. All the hard and soft constraints pose some challenge to the implementation of this system.

The problem consists of assigning to each course the teachers responsible for teaching the various classes (theoretical, practical, etc.). This is done while trying to respect the preferences of the teachers regarding the courses to teach, the maximum, minimum and desirable hours to be assigned to each teacher in each academic year.

The solution can be separated into several stages: First, the preferences of the teachers and of some administration members are initially collected and checked. This is followed by the generation of the assignment between teachers and course sections. The final stage consists of some manual adjustments to the generated solution, if necessary.

Chapter 3

Literature Review

3.1 Introduction

The Teacher Assignment Problem, addressed herein, can be considered a resource scheduling problem. This type of problem consists of, given a finite set of resources and a set of tasks as well as a certain number of constraints, finding the best way to assign the resources to the tasks such that the objective is optimised.

Timetabling problems, and University Timetabling in particular, are a pretty well researched area. However, from the sub-problems defined by Carter and Laporte [1998], the Teacher Assignment Problem can be considered one of the least studied ones [Domenech and Lusa, 2016]. It has, however, been studied in some detail over the last fifty years and one of the first solutions to the problem was presented by Tillett [1975], where the problem was solved using a binary integer programming approach in a high school context. The work by Hultberg and Cardoso [1997] has shown this problem can be considered a special case of the fixed charged transportation problem and that it is a NP-hard problem. While in a classical transportation problem the cost is proportional to the number of transported units from a producer to a consumer, in a fixed charge transportation problem there is also a fixed starting cost associated to each producer-consumer pair.

While most work done regarding this problem is for assigning the teachers before the courses are scheduled, some of it is done for assigning the teachers afterwards [Domenech and Lusa, 2016]. In that case the assignments have to take into consideration the schedules, for instance, a teacher can not teach two classes which have overlapping schedules. However, for the case where the assignment is done beforehand, one might need to take into consideration certain factors which might influence the course scheduling which is done afterwards.

3.2 Related Work

Due to computational limitations faced when dealing with large instances of the problem, implementations in the literature have usually resorted to heuristics and meta-heuristics such as simu-

lated annealing [Gunawan and Ng, 2011], genetic algorithms [Wang, 2002, Qin et al., 2016] and tabu search [Gunawan and Ng, 2011]. However, with the latest developments in both hardware and operations research software, integer programming and mixed integer linear programming models have started to be increasingly used to solve some University Timetabling Problems, obtaining in some cases solutions which are optimal or near-optimal [Domenech and Lusa, 2016]. Hmer and Mouhoub [2010] have shown that it is possible to solve a variant of the Teacher Assignment Problem, where the courses are already assigned to time slots, using Constraint Programming.

Some research has also been done for solutions combining multiple of the sub-problems in the classification done by Carter and Laporte [1998]. One example is the combination of both Teacher Assignment and Course Timetabling problems [Gunawan et al., 2007]. This combination consists of assigning teachers to courses and time slots at the same time. In some cases the Classroom Assignment is also combined into the other two problems [Avella and Vasil'Ev, 2005]. While this fusion has the potential to provide better solutions, it makes the problem to be solved much more complex. This is a problem specially for large instances. For this reason timetabling problems can be split into multiple phases [Gunawan and Ng, 2011].

Andrew and Collins [1971] are responsible for one of the first models for solving the teacher assignment problem. The problem was modelled with a linear programming approach. The objective function consisted of the weighted sum of the teachers' preferences and their effectiveness as evaluated by the department. The used constraints guaranteed that each course was taught and that each teacher had a certain workload.

In the work developed by Tillett [1975], a model for a high school teacher assignment problem was developed using a binary integer programming approach. This model took into account the preferences of the teachers to each course, expressed in a 1 to 9 Likert scale and the teachers were assigned to course sections. This model allowed the preference of a teacher for a course to vary according to the number of sections of the course. It also had constraints for the maximum number of unique courses per teacher. The largest department the work was applied had 13 teachers and 14 different courses. The author concluded that the execution of the algorithm was prohibitively expensive in terms of computational time with the resources available back then.

Below is the review of some of the literature regarding the teacher assignment problem. Unless stated otherwise, each work is related to the assignment of teachers before the course scheduling.

Breslaw [1976] approached and tried to overcome the major problem in the model proposed by Tillett [1975], the computational time. By modelling the problem as a transportation problem solved by a linear programming model, the computational power needed is greatly reduced. The newly proposed model could be now applied at the university level. However, this model is applied to a case where the courses scheduling already exists. The used objective function maximised the teachers preferences and the Revised Simplex algorithm was used to solve the model which was tested with a data set of 120 courses and 27 teachers. An optimal solution was obtained in just 13 seconds. This efficiency is obtained by the fact that the problem is modelled as a simple transportation problem and which is solved by the Revised Simplex algorithm. This model however, does not take into account the number of courses assigned to each teacher and the workload bal-

ance. It only maximises the teachers' preferences in relation to courses' time-slots and forces the work load of each teacher to be equal to a predetermined number.

Dyer and Mulvey [1976] formulated the problem as a network model solved by a network optimisation algorithm. The author states that although some useful details could not be captured by this type of model, the lower cost of solving the model outweighed the omitted detail and allowed the scheduler to use the model more frequently and as an iterative solution strategy. In order to handle cases in which the teachers were not enough or when the teachers did not have enough to teach, the network had an extra virtual teacher and course for each curricular area. Those had low preference values in order for them to be used only when necessary. For a given area, the virtual teacher could be assigned to any remaining course and the virtual course could be assigned to any teacher without enough courses to teach.

McClure and Wells [1984] describe an integer programming model in which the teachers are assigned to course sections unlike previous models where the teachers were assigned to whole courses. The teachers preferences are related to the courses' schedules, not just the courses themselves. This means the schedules for the courses should already be generated. The model was solved using the LINGO software package. It was tested with a data set of 18 teachers which was reduced to only 6 due to there being no overlap in the schedules selected by the other teachers.

Partovi and Arinze [1995] consider the problem of assigning teachers to courses at a university and solve it using a knowledge-based system (KBS) approach. In this type of approach the system tries to replicate the decisions made by humans, mostly by if-then rules. The system was developed in Prolog and tested with a data set consisting of 19 courses and 12 teachers. They considered the results as satisfactory and very similar to the ones generated by the human faculty schedulers.

Hultberg and Cardoso [1997] developed a model for assigning teachers to classes while minimising the distinct courses assigned to each one. The problem was formulated as a mixed integer linear program and a branch and bound solution was developed and compared to a solution developed with the commercial solver CPLEX. Four departments were tested being that the largest one had 30 subjects to be taught by 54 teachers. Other bigger random data sets were also generated and tested. For the real test cases, both algorithms found similar or better solutions than those which had been created manually. The branch and bound algorithm developed by the authors managed to return solutions in less than half a second for cases where CPLEX took up to 598 seconds, while still providing a better solution than CPLEX. While the algorithm did not take into account the fact that not all teachers can or will teach any subject, the authors considered the results impressive and important for the solving of teacher assignment problems of higher size.

Al-Yakoob and Sherali [2006] developed two integer programming models for solving the teacher assignment problem with already provided course schedules. This model was applied at Kuwait University. It had some constraints such as having a limit of 15% for changes to the timetable and a gender-based policy imposed by law. The first model assigned teachers to the classes to be taught based on the minimisation of the total dissatisfaction and the sum of the differences in dissatisfaction between teachers. The second model is derived from the first and aims to further increase the teachers' satisfaction by allowing up to 15% of changes to the previously

generated timetable. This model has additional components in the objective function, in order to keep the quality of the schedules. This includes spreading the classes over the available time-slots, in order to provide alternatives to students, and avoiding class conflicts so that students can simultaneously take any two of such classes. The models were solved using CPLEX-MIP 7.5. Six different test cases were used, with these containing up to 30 different courses, 94 sections and 37 teachers. The test cases were run until a out-of-memory error was encountered. The best result was that of a solution which was within 50% of optimality, with this result taking 48 hours to be obtained. Due to the fact that this model had not not provided very satisfactory results, a specialised LP-based heuristic was developed. This heuristic iteratively enforced the integrality of only a subset of the integer variables. The author considered this heuristic to provide good results for all the test cases in terms of optimality percentages, providing a solution within 68% of optimality for the test case which had obtained 50% without the heuristic.

Gunawan et al. [2007] used an hybrid algorithm which combines an integer programming approach, a greedy heuristic and a modified simulated annealing algorithm for solving both the teacher assignment and the course scheduling problems simultaneously. The algorithm was compared to a solution based only on CPLEX (OPL Studio). The hybrid algorithm had much lower execution times but the score of the results was in general poorer than those provided by CPLEX. The algorithms were tested with data sets with dimensions up to 30 teachers and 60 courses.

Gunawan et al. [2008] implemented a genetic algorithm for solving the teacher assignment problem. The algorithm consisted of two stages. The first stage consisted of assigning teachers to courses. In the second stage the teachers were then assigned to the courses' sections while trying to balance the teachers' load. The largest tested data set was artificially generated and had 100 full time teachers and 200 courses. The algorithm's results were evaluated by testing it against two real data sets. The authors concluded the algorithm yielded better results than those which had been created manually.

Hmer and Mouhoub [2010] solved the teacher assignment problem using a constraint programming approach and a solver implemented by the authors. This solver was implemented in C# and based on "Java Cream" solver. In this re-implementation, the authors modified the backtracking algorithm to take into account not only the hard constraints, but also the soft ones. A web-based application was also developed in order to get the teachers to enter their preferences into the system. The algorithm was only tested with a data set of 17 courses and 10 teachers.

Gunawan and Ng [2011] developed two meta-heuristics for solving the teacher assignment problem: simulated annealing and tabu search. Similarly to the work done by the authors in 2008, the algorithms consisted of two stages. In the executed experiments, Tabu search had the best results and both algorithms outperformed the previous implementation, using a genetic algorithm, in terms of the minimisation of the workload variance between teachers. The largest tested data set had 100 full time teachers and 200 courses. The authors concluded that both algorithms were able to generate good solutions to the problem.

Wilson et al. [2013] also used a genetic algorithm approach to tackle and solve the teacher assignment problem. The algorithm used a objective score with multiple weighted components

and was tested with 167 courses and 45 teachers. The results were obtained in a question of minutes and the authors considered them appropriate.

Moreira and Reis [2013] formulated the problem as a multi-agent system where the teachers are represented by computational agents that can cooperate by forming alliance groups. In this system, each agent would have utility values for joining the coalition which teaches each course, for belonging to a coalition with certain teachers and for the total number of coalitions that each teacher belongs to. The implementation was left as possible future work.

Thipwiwatpotjana [2014] developed a Mixed Integer Linear Programming model where the teachers' desired workload is expressed in terms of intervals instead of single values. IBM ILOG CPLEX Optimizer was used for solving the model and it was tested with 58 teachers and 98 courses.

Domenech and Lusa [2016] developed a different Mixed Integer Linear Programming model for the teacher assignment problem with the courses' schedules already generated. This model balanced the teachers' teaching load while maximising the teachers' preferences for courses. IBM ILOG CPLEX Optimizer was used for solving it. The model was tested with data sets with up to 50 teachers and 200 courses. The authors considered the solutions were good for data sets with up to 40 teachers.

Besides these solutions, some different ones have been used for the other related sub-problems, such as neural networks: Carrasco and Pato [2004] applied a neural network solution for solving the class-teacher timetabling problem at a Portuguese university, having obtained better results than a previously used genetic algorithms approach [Carrasco and Pato, 2001]. This problem, which comes after the teacher assignment problems, consists of scheduling the class-teacher meetings without conflicts. The largest tested instance had 107 teachers, 92 classes, 27 rooms and 50 time periods.

It is clear that most of the recent work using mathematical programming approaches have relied on IBM ILOG CPLEX. According to benchmarks by Hans Mittelmann [2018], in this case for mixed integer linear programming, the major players in this market are Gurobi [1], CPLEX[2] and XPRESS[3], which are relatively close to each other in terms of average performance, with Gurobi in first place followed by CPLEX. These three software packages outperform the other available ones by a large margin, as can be seen in Fig. 3.1.

While integer and linear programming approaches have been rather researched, constraint programming is one approach which has had limited focus. According to some literature on other problems, for some cases constraint programming is capable of providing solutions of much better quality than integer and linear programming approaches, while being simultaneously easier to model [Laborie and Messaoudi, 2017]. However, in other cases, the results can be worse [Trilling et al., 2006].

[1] More information available at: http://www.gurobi.com/

[2] More information available at: http://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer

[3] More information available at: http://www.fico.com/en/products/fico-xpress-solver

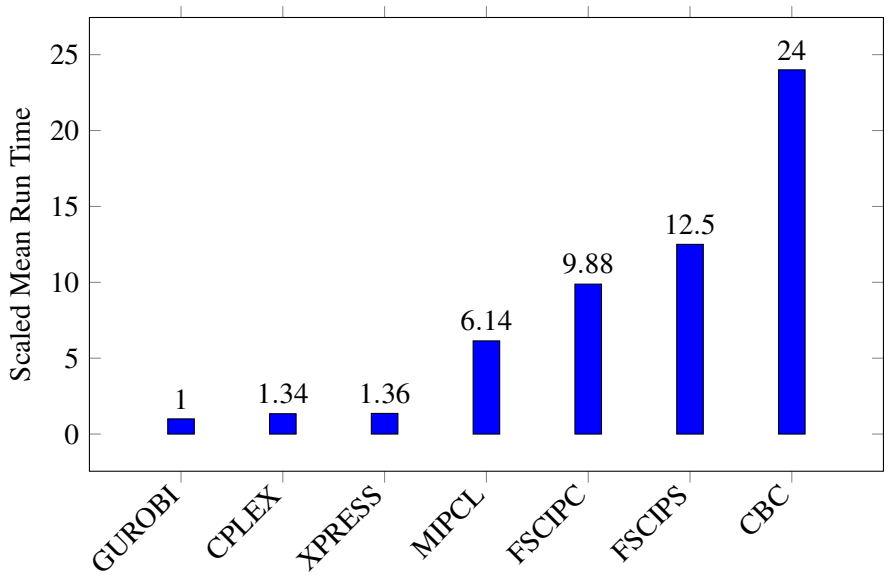

Figure 3.1: Mixed Integer Linear Programming Software Benchmarks [Hans Mittelmann, 2018]

In terms of constraint programming software, it seems that there are no relevant benchmarks available for comparing the multiple available alternatives. The work by [Laborie, 2009] shows that IBM ILOG CP Optimizer, with its automatic search and default parameters, is able to outperform state-of-the-art solutions for the three problems tested. According to the author, this is consistent with their experience of using this software to solve industrial scheduling problems. Popular alternatives to CP Optimizer include, but are not limited to, Gecode, Choco, Comet, JaCoP and Minion [Nordlander, 2009].

Richter et al. [2008] developed a constraint programming model for assigning skilled professionals to high-end positions at a large organisation, in this case IBM services organisations. There is a certain similarity between this and the teacher assignment problem. In this problem, known as workforce management, non-accurate assignments can result in large monetary losses to the business. According to the authors, traditional operations research methods are not capable of solving this problem, unlike constraint programming. IBM CPLEX CP Optimizer was used and a new constraint propagator was implemented in order to efficiently model some of the required constraints. Furthermore, the authors were able to use the same model for generating a new assignment based on a previous one where some positions were added and removed and some employees left while others joined. The idea is to obtain a near-optimal solution which keeps the matches as close as possible to the previous ones. This is known as a flexibility problem in the literature. While the most common approach is based on stochastic local methods, the authors used the same systematic solver in order to simplify the code and to make the implementation work properly with large changes. This was done by employing two different heuristics: variable ordering and value ordering. The experiments made show that the both the original and the flexibility problems were solved efficiently. The used real data set had 4000 employees and 1354 positions. It was

partitioned in two parts, a main one containing 3200 employees and 1083 positions, and another used for the flexibility experiments with the remaining data. The original problem was solved in only 8 seconds and produced 263 assignments. The authors considered this latter number appropriate since only good quality matches were being accepted. For the flexibility experiments four levels were tried: 1%, 5%, 10% and 25%. For each percentage, the correspondent quantity of data was deleted and the same amounts of positions and employees were added from the smaller data set partition. The results show that the solutions are obtained in similar execution times and that the score is similar to the original without many changes to the existing matches. Further developments to this work are presented in Asaf et al. [2010].

3.3 Summary and Conclusions

Table 3.1: Literature comparison

Authors	**Approach**	**Pre-Timetable**	**Preferences**	**Other Features**
Andrew and Collins [1971]	LP	Yes	Yes	
Tillett [1975]	IP	Yes	Yes	Limits the number of unique courses per teacher
Breslaw [1976]	LP	No	Yes	Limits the number of unique courses per teacher
Dyer and Mulvey [1976]	Network Flow	Yes	Yes	
McClure and Wells [1984]	IP	Yes	Yes	
Partovi and Arinze [1995]	KBS	Yes	Yes	
Hultberg and Cardoso [1997]	MILP / B&B	Yes	No	Minimises the number of unique courses per teacher
Al-Yakoob and Sherali [2006]	MILP	No	Yes	
Gunawan et al. [2007]	Hybrid (IP + heuristics)	Yes	Yes	Combines teacher assignment with course scheduling
Gunawan et al. [2008]	Genetic Algorithms	Yes	No	Minimises the teachers' workload variance
Hmer and Mouhoub [2010]	CP	No	Yes	Limits the number of unique courses per teacher
Gunawan and Ng [2011]	Tabu Search Simulated Annealing	Yes	No	Minimises teachers' workload variance
Wilson et al. [2013]	Genetic Algorithms	Yes	No	Maximises teacher expertise in the assigned courses
Thipwiwatpotjana [2014]	MILP	Yes	Yes	Desired workloads stated as intervals
Domenech and Lusa [2016]	MILP	No	Yes	Minimises teachers' workload variance Maximises teachers' preferences

Table 3.1 shows a comparison between the different implementations for solving the teacher assignment problem which were analysed and described in this chapter.

Although there are has been some research into the teacher assignment problem, there are some topics which have not yet been explored, such as the flexibility aspect. This aspect can be explored not only in the fact that the teacher-section assignments should not change much from one year to the following, but also that teachers should preferably stay in a course for a set minimum number of years.

Furthermore, most implementations have trouble dealing with large data sets, such as ones involving more than 40 teachers. This problem can manifest itself either as a long run time or in poor results.

In addition to the above, some possible preferences have not yet been explored, such as preferences towards other teachers and the fact that the administration might want to create special types of constraints, such as forcing that one teacher should be assigned to a specific course or that that teacher should at least have a higher weight for that assignment. Furthermore, the fact that some course sections can be shared between different classes or even courses has not been explored either.

Finally, in cases where one problem depends on the solution of the previous, one should take care to provide solutions which take in mind the following problem. In this case, the teacher assignment problem should provide solutions which take into account the generation of the courses' timetables, which is done afterwards. For example, a course with many classes should not have them all assigned to the same teacher as this would result in classes not being able to take place in parallel.

Chapter 4

Planning

The purpose of this chapter is to present the planned approach for the achievement of the objectives of this dissertation.

4.1 Current Process

In order to better understand the context from where this project emerged and how it fits into the current process, in this section this process is presented.

The current process for solving the teacher assignment problem at the department consists of the following steps:

1. An email is sent to the programme directors asking them to send the course requirements.
2. The requirements from the previous step are received.
3. The scientific area coordinators send their requests for the lessons requirements and any pre-assigned teachers.
4. The assignment matrix is manually created.
5. The matrix is analysed and corrections are made.
6. The teacher assignments are proposed.
7. Necessary guest teachers are hired.
8. The final assignments map is released.

The objective of this project is to receive the data from step 2, teachers are asked to provide their preferences during step 3, the data from step 3 is inserted into the system at the end of that step and, finally, the system replaces step 4 by generating the assignments automatically. All the other steps remain the same.

4.2 Approach

The planned approach consisted of the following steps: In order to easily get the teachers' preferences and to allow an easy administration of the system, a website is built for these purposes. Afterwards, the teachers are asked to input their preferences into it and at the same time, a system for solving the teacher assignment taking into consideration the data collected from the website is implemented. The final step is the testing of the whole system by using the teachers' preferences which were collected as well as the data provided by SIGARRA.

For the solver, two different approaches were planned to be tried. One using mixed integer linear programming and another with constraint programming. The former has been more thoroughly explored in the literature while the latter has had little attention. Both solutions were to be compared in order to find the most adequate and performant one. However, in the end, only the constraint programming one was developed due to time constraints.

A solver for this problem must take into account the teachers' preferences and the administration defined assignments. Moreover, it should minimise the changes in terms of assignments from one year to the next. Finally, the system should also try to keep teachers in their previously assigned courses for a certain number of years in order to avoid them having to prepare new courses every year.

In addition to what was stated above, some additional scripts were also needed in order to parse some files needed for the importing of the required data.

Teachers can provide preferences not only for courses and the respective lesson types, but also preferences in relation to other teachers, which they might want to work with or not.

In addition to these preferences, information about how the teaching load should be balanced across both semesters can also be provided.

Finally, the system has administrative users, who can add, edit and remove teachers, review some of the collected information, start or stop the assignment solver and adjust the final result manually. The information about the teachers preferences towards other teachers is private.

4.3 User Stories

In this section the user stories representing the implementation plan are presented.

- As a visitor, I want to be able to login into the system in order to identify myself and to gain access to system's functionality.

- As a teacher, I want to be able to state how I would prefer my teaching load to be distributed across both semesters, so that this information can be taken into account. This should be input as two values, each representing the desired load for each semester. The sum should equal the required total value, as shown in Fig. 4.1.

Figure 4.1: Workload Preferences Interface

- As a teacher, I want to be able to state how I rate each course according to my preference so that this information can be taken into account. This should be represented as a scale from -N to N for each course, as shown in Fig. 4.2.

Figure 4.2: Course Preferences Interface

- As a teacher, I want to be able to rate each teacher according to my preference, so that this information can be taken into account. This should be also represented as a scale from -N to N for each course, as shown in Fig. 4.3.

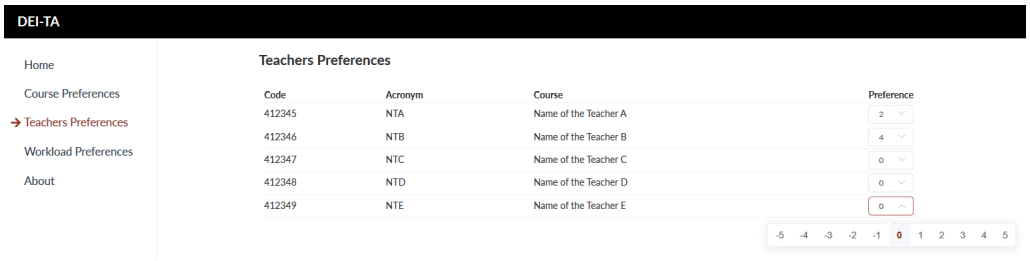

Figure 4.3: Teachers Preferences Interface

- As an administrator, I want to be able to import the courses' and teachers' information from SIGARRA, including sabbatical leaves and workload reductions, so that the system has updated information.

- As an administrator, I want to be able to link two lesson requirements for when lessons are shared between courses, so that the system has more information for generating better assignments.
- As an administrator, I want to be able to manually assign teachers to certain courses with a certain weight, so that the generated assignment follows certain requirements.
- As an administrator, I want to be able to review some of the information input by the teachers, in order to check the status of the preferences collection.
- As an administrator, I want the system to be able to automatically generate the teacher-course assignments in order to solve the teacher assignment problem at DEI.
- As an administrator, I want to be able to configure the weights used by the solver, so that I can fine-tune it to provide more adequate results.
- As an administrator, I want to be able to start and stop the solver, in order manage when it should start and eventually stop before finishing if necessary.
- As an administrator, I want to be able to check the status of the solver and to check the final result when available, so that I can easily keep track of the progress and view the solution.
- As an administrator, I want to be able to manually change the assignment results, so that I can fix potential problems in the automatically generated one.
- As an administrator, I want to be able to import the assignment results into SIGARRA, so that the solution can be used.

Figure 4.4 shows the use case diagram for the proposed solution.

4.4 Planning and Risk Analysis

4.4.1 Planning

Taking into account the requirements of the system and the time available for its execution, a set of adequate activities was determined and scheduled. In addition, possible foreseen risks are studied as well as possible ways to mitigate them.

The Gantt chart with the planned tasks which were to be completed during this dissertation is presented in Fig. 4.5.

During the first semester, the problem is defined, an analysis of the state of the art is performed, the implementation requirements are gathered and the initial report is written.

During the second semester, the first step is the implementation of the platform for collecting the teachers' preferences. This consists of the front-end used by the teachers for providing their preferences and the part of the back-end required for receiving and storing that information in the database.

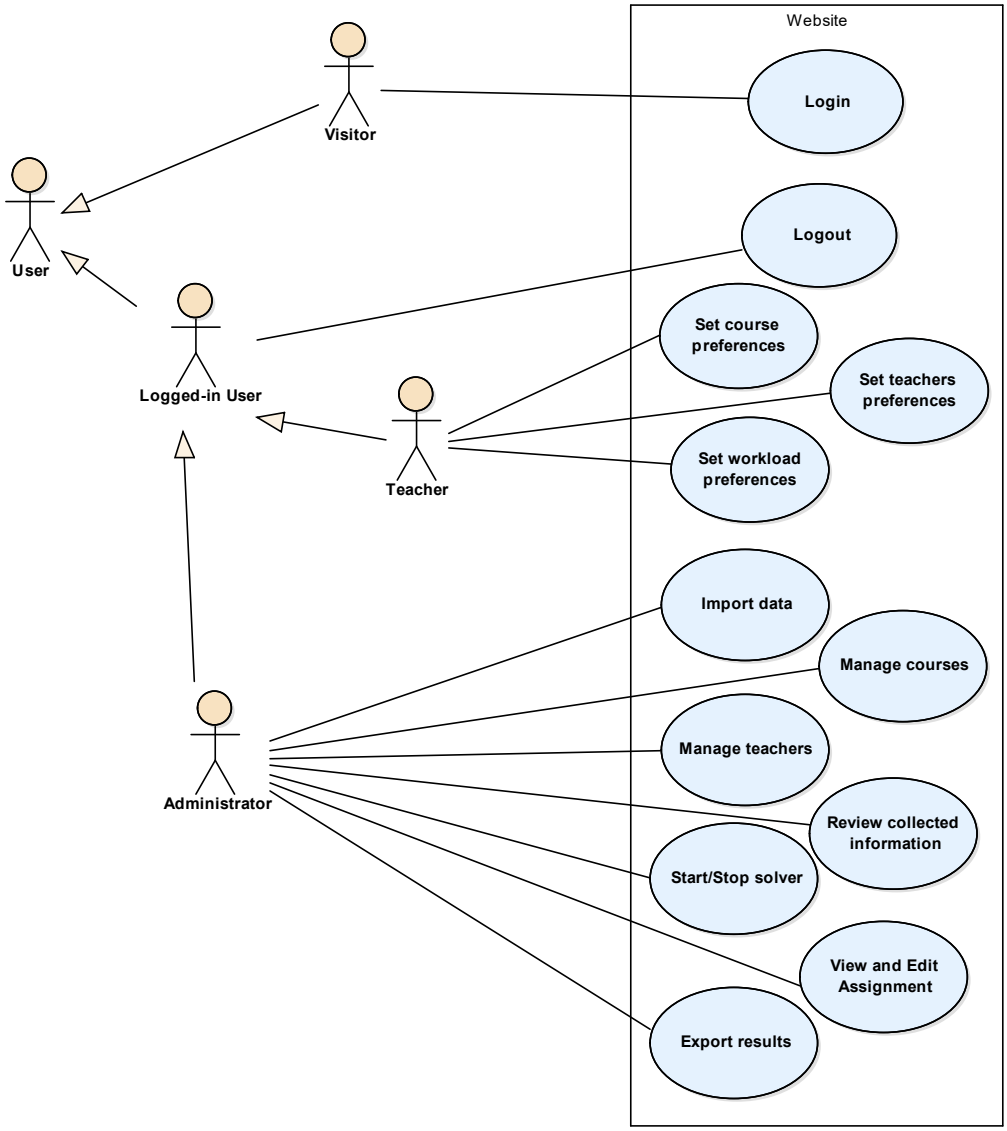

Figure 4.4: Use case diagram for the proposed solution

Afterwards, the technologies needed for the implementation of the solver are studied, in order to avoid mistakes caused by the misunderstanding of the technologies.

Subsequently, the solver is implemented including some tests with dummy data. This data includes small test cases as well as the data provided by SIGARRA using random preferences for each teacher.

Then, the module for doing manual adjustments to the solution is developed. This consists of a matrix for manually changing the results of the solver, which are the assigned hours and the respective factor between courses and teachers.

Afterwards, some end-to-end testing of the whole system is performed, including tests with real data for the following academic year. A analysis of the results is done, with metrics for comparing the results with those of a manual assignment.

ID	Task Name	Duration	Start	Finish
1	**Planning**	**18 wks**	**Mon 09/10/17**	**Fri 09/02/18**
2	Problem Definition	4 wks	Mon 09/10/17	Fri 03/11/17
3	Literature Review	18 wks	Mon 09/10/17	Fri 09/02/18
4	Gathering of the implementation requirements	9 wks	Mon 09/10/17	Fri 08/12/17
5	Writing of the report	13 wks	Mon 13/11/17	Fri 09/02/18
6	**Dissertation**	**19 wks**	**Mon 12/02/18**	**Fri 22/06/18**
7	Implementation of the platform for collecting the teachers' preferences and other information	2 wks	Mon 12/02/18	Fri 23/02/18
8	Study of CPLEX Optimizer and CP Optimizer	2 wks	Mon 26/02/18	Fri 09/03/18
9	Implementation of both solvers	8 wks	Mon 26/02/18	Fri 20/04/18
10	Implementation of the manual adjustments module	4 wks	Mon 23/04/18	Fri 18/05/18
11	End-to-end testing	2 wks	Mon 21/05/18	Fri 01/06/18
12	Writing of the dissertation	19 wks	Mon 12/02/18	Fri 22/06/18

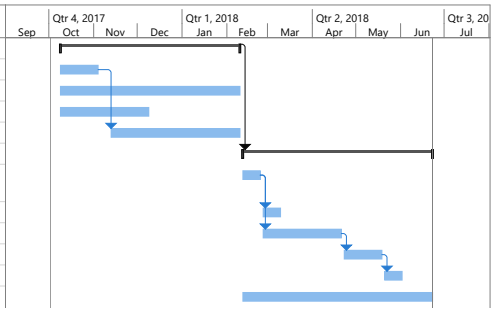

Figure 4.5: Plan for the tasks represented by a Gantt Chart

Finally, the writing of the dissertation is finalised, taking into account the the development and analysis which were done for this project.

4.4.2 Risk Analysis

Two types of risks were considered, technological and human, as well as three levels of probability and impact: high, medium and low. There were only a small number of foreseen risks, with them being shown in Table 4.1.

Table 4.1: Various risks predicted for the development of this dissertation

Type of Risk	Risk	Probability	Impact
Technological	Impossibility of generation of a quality solution	Medium	High
Human	Delays in delivery of the required input data	Low	High
Human	Delays in the implementation of the web platform	Low	Medium
Human	Delays in the implementation of the solver	Medium	High
Human	Delays in the filling of the teachers' preferences	Medium	Low

From these, only three were considered more dangerous. The first one is related to delays in the delivery of the required input data, which comes from SIGARRA. Another one of them is the impossibility of generation of a quality solution in a reasonable time due to the dimension of the problem and limitations of the current state of the art in the used software for the creation of the solver as well as of the current computational power available in modern hardware. The final one is the possibility of delays in the implementation of the solver which due to being the most important part of the project can have a very serious impact.

As a way of mitigating these last two problems, the tools used for the creation of the solver were planned be studied very well beforehand. This was to allow familiarity with the way of solving such problems in order to reduce both the likelihood of the implementation not being able to solve the problem in a reasonable time as well as possible delays in its implementation. For the

problem related to the data, in case of a delay, dummy data was to be used until the real one was delivered.

4.4.3 Actual Work Overview

Several factors contributed to the delay in the original plan and culminated in the decision to develop only one of the models. It was decided that the constraint programming approach was to be followed due the fact that it was much less explored in the literature compared to integer and linear programming approaches and because some related work had good results with this chosen approach [Richter et al., 2008].

The main reasons for this delay were due to the poor quality of the CPLEX documentation, which made the learning and usage of CPLEX CP Optimizer more difficult than anticipated, as well as a higher complexity for the implementation of the required model than previously expected. Delays in the delivery of the required data as well as some problems with its quality also contributed to these delays, as dummy data had to be created and some changes had to be made to the system when the data was finally delivered, as some of the data structure was different from what had been anticipated.

4.5 Technological Choices

In terms of technological choices, some decisions had to be made considering the project requirements.

Since the database schema is highly relational, a relational database was the best option. In order to avoid unnecessary commercial licenses, it was agreed an open-source database would be selected. The most popular options that fit these requirements are MySQL[1] and PostgreSQL[2] [StackOverflow, 2018]. While MySQL is known to have better performance for some use cases, database performance is not of the utmost importance for this project. PostgreSQL was chosen for its higher number of features, which helped ensure the database was not going to be a limitation in the implementation. For example, its native support for the JSON and JSONB formats allows for a part of the data, which is not relational, to be stored in a non-relational way. This helps to reduce the database schema complexity, while still allowing to easily query and modify those data. PostgreSQL also better follows the SQL standard and has a better guarantee of data integrity [DigitalOcean™ Inc, 2014].

For the website front-end, in order to provide a richer and more interactive user experience a JavaScript framework was chosen. Most popular options include React[3], Angular[4] and Vue.js[5]. These follow the Model-View-ViewModel pattern (MVVM), which allows for an easier development and maintainability of the project [Microsoft, 2012]. From these, Vue.js was chosen mainly

[1] More information available at: https://www.mysql.com
[2] More information available at: https://www.postgresql.org
[3] More information available at: https://reactjs.org/
[4] More information available at: https://angular.io/
[5] More information available at: https://vuejs.org/

due to personal preference since all of the options are comparable [Vue.js, 2016]. As for the back-end API, in order to use a single programming language for both ends, Node.js[6] was chosen for its development. Node.js allows for fast development due to its dynamic nature while still outperforming other dynamic languages such as Python and PHP by a large margin [Lei et al., 2015].

In terms of the solver, IBM ILOG CPLEX was chosen since it supports both integer linear programming and constraint programming and performs well. In addition, CPLEX improves significantly with each release version of the software [IBM, 2018]. CPLEX Optimizer supports mixed integer linear programming while CPLEX CP Optimizer supports constraint programming.

The two components mentioned above have three different library platforms in common: C++, Java and .NET.

CPLEX's Java API was chosen for the implementation of the TAP solver and the respective server in Java, since it is the only high-level and cross-platform language option provided by CPLEX. The .NET library uses a DLL and is available for Windows only [IBM, 2017].

Finally, Docker containers were chosen to be used in order to make the project easier to deploy and maintain. This is due to the fact that with Docker, no dependencies have to be installed on the machine except for Docker itself.

4.6 Summary and Conclusions

According to the plan, a web app for collecting the teachers' preferences was developed, followed by the implementation of a solver using constraint programming. Afterwards, a manual adjustments module was implemented, in order to allow for manual adjustments to the automatically generated solution, and the system was tested with both real and artificial data.

[6]Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine (`https://nodejs.org`).

Chapter 5

Implementation

In the current chapter, the work which was developed for this project is presented. This is done with an overview of the whole system architecture and by going into detail about each of its components. The inputs and outputs of the system are also explained as well as some details about its deployment.

5.1 System Architecture Overview

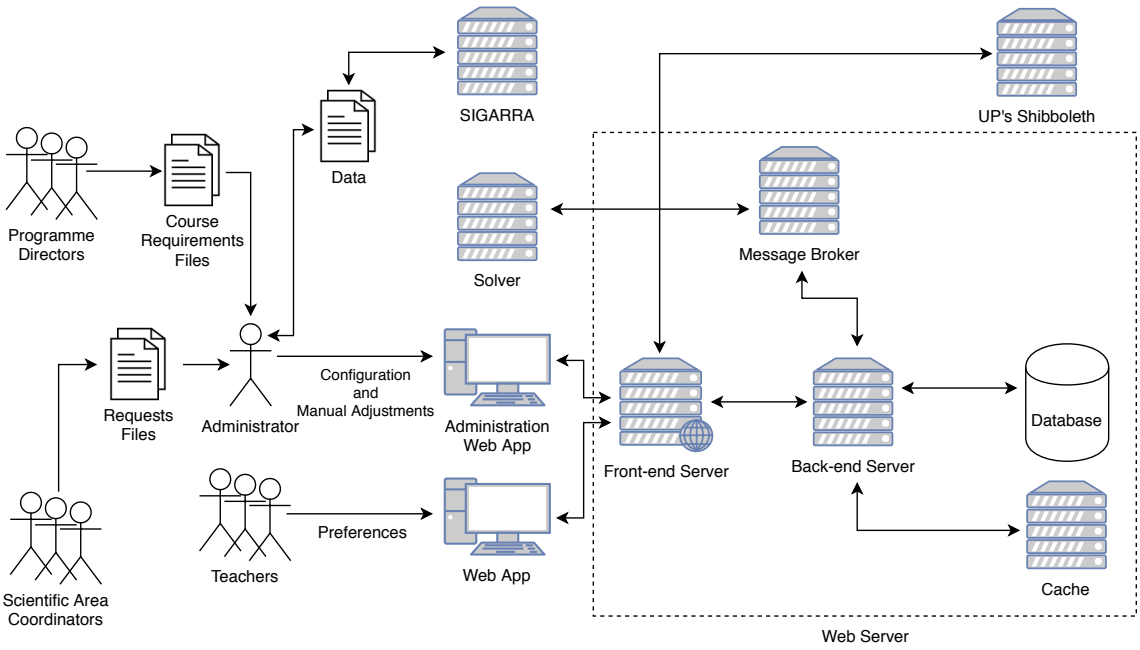

Figure 5.1: System Architecture

The implemented system consists of six different components, shown in Fig. 5.1: a Front-end Server, a Back-end Server, a Database, a Message Broker, a Cache Database, and finally, the Solver.

This decoupled architecture was followed because it allows for the solver to be on a separate machine from the rest of the system. This allows the web server to have lower system requirements as the solver requires more CPU and RAM resources. It also allows the system to continue working even if the solver is not on the network. When the solver is turned on it connects to the message broker and immediately starts processing pending tasks. This architecture also allows for multiple solvers, which can be useful if the administrator wants to try different solver inputs in parallel.

In addition, the system also interacts with UP's Shibboleth[1], a Single Sign-On Authentication system used by the university. It also receives and exports data from and to SIGARRA, information which is currently transferred through CSV ou Excel files.

All these components, except for the Solver, are containerised with Docker and the whole system deployment is done with Docker Compose[2]. This makes the system easy to deploy, despite the fact of having several components, to the point where it can be started with just a single command. The Solver is deployed as a Java JAR file and is run on a different machine. It only has CPLEX and Java as dependencies.

The front-end consists of a Nginx[3] reverse-proxy which serves the front-end static files, which were developed with Vue.js, while also proxying the API calls to the back-end server. The interface for creating solver tasks uses websockets in order to show the status of the process in real-time. This interface is shown in Fig. 5.2.

The back-end server was developed with Node.js and it communicates with all other systems. For authenticating the teachers, it integrates with the university's Shibboleth. This service uses the SAML protocol in order to authenticate the users via a Single Sign-On page.

The message broker is a RabbitMQ[4] instance. It allows to easily use the publish/subscribe pattern, which allows in this case for the solver to be notified when there is work to be done, and for the back-end to be notified by the solver when progress occurs in one of the tasks.

The used database system was PostgreSQL, for the reasons mentioned previously in Chapter 4, which were mainly due to the relational nature of the data.

A Redis[5] instance was used to store authentication information and to synchronise web socket broadcasts between back-end nodes. This makes the back-end stateless, which allows for it to be run with more than one process, which in Node.js is known as cluster mode. This improves the scalability and availability of the service. The synchronisation is necessary as events happening on one node have to sent to the other nodes so that the information to be sent through websockets can be done independently from which node the user is connected to. The Redis instance could have also been easily used for caching certain API requests or database query results, should that have been necessary.

[1] Shibboleth is a federated identity solution (https://www.shibboleth.net/)

[2] Compose is a tool for defining and running multi-container Docker applications (https://docs.docker.com/compose/)

[3] NGINX is an HTTP and reverse proxy server (https://www.nginx.com/)

[4] RabbitMQ is an open source message broker software (https://www.rabbitmq.com/)

[5] Redis is an open source in-memory data structure store (https://redis.io/)

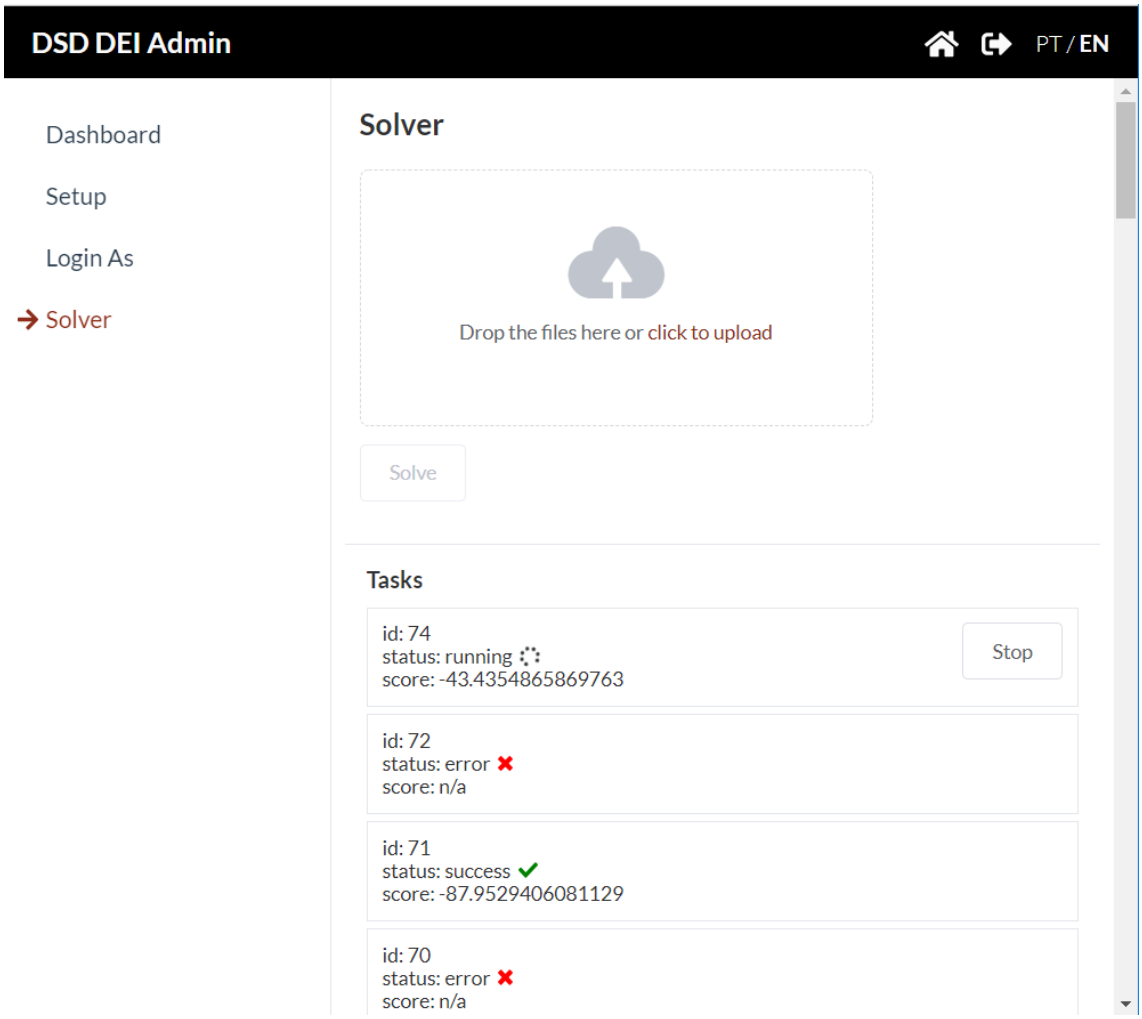

Figure 5.2: UI for the solver

Finally, the solver is just a Java JAR file which can be run on any system on the network and which has both Java and CPLEX installed. Next to the JAR file should be a file which has both the hostname and the password for the message broker so that it can connect to it.

5.2 Inputs and Outputs

Several types of inputs are used by the system. The data provided by SIGARRA, the course requirements handed over by the programme directors, the teachers' preferences, the requests made by the scientific area coordinators and the manual input provided by the administrator.

The input from SIGARRA includes the list of teachers and previous years assignments. The teachers information includes an identification code, name, rank, department, information about being from a different college, the target number of hours to be taught per week as well as the scientific areas, information about a possible unavailability in a certain semester due to a sabbatical leave and additional roles which the teacher might have, such as being the director of a department.

The previous year assignments include the following relevant information: teacher code, year, course code, type of lesson, assigned hours and factor.

The course requirements, provided by the programme directors, include a list of the courses which are to be taught in the year which is being planned. This information includes course code, academic year, semester, code, acronym, name and scientific areas.

The requests made by the scientific area coordinators include for each course the types of the required lessons and for each type the number of hours per lesson as well as the required number of lessons. From this information it is possible to calculate the number of hours per type of lesson and the total number of hours per course.

Some teachers are from different departments and teach some of DEI's courses while in other cases, teachers from DEI teach courses from other departments. The input includes all of these teachers and courses and they must be differentiated by the solver.

In addition to the above, a few courses from different programmes have some classes in common between them. This information is also provided as a link between two courses with an associated percentage which represented the ratio of classes which are shared.

Finally, the input provided by the administrator can include some type of changes to the available information as well as manual adjustments to the final solution.

The output, for exporting into SIGARRA, consists of a list of assignments similar to the imported list of previous assignments. Each row includes the teacher's code, course's code, type of class, number of hours assigned and the respective factor.

5.3 Database

The database is responsible for storing all the data required by the system. The final database diagram is presented in Fig. 5.3. Below is a description of each class in the database:

- **Admin**: Represents an administration account with a username and hashed password.
- **Instance**: An instance represents an academic year where the system is set up, as the system is instantiated for each one.
- **Course**: Represents a course, associated to an instance.
- **Teacher**: Represents a teacher, associated to an instance.
- **Rank**: Represents a teacher's rank.
- **Settings**: Used for storing global settings, such as the instance id of the current active instance. This table only has a single row.
- **Request**: Represents a request for a solution, consisting of a list of lesson requirements and some associated pre-assignments.

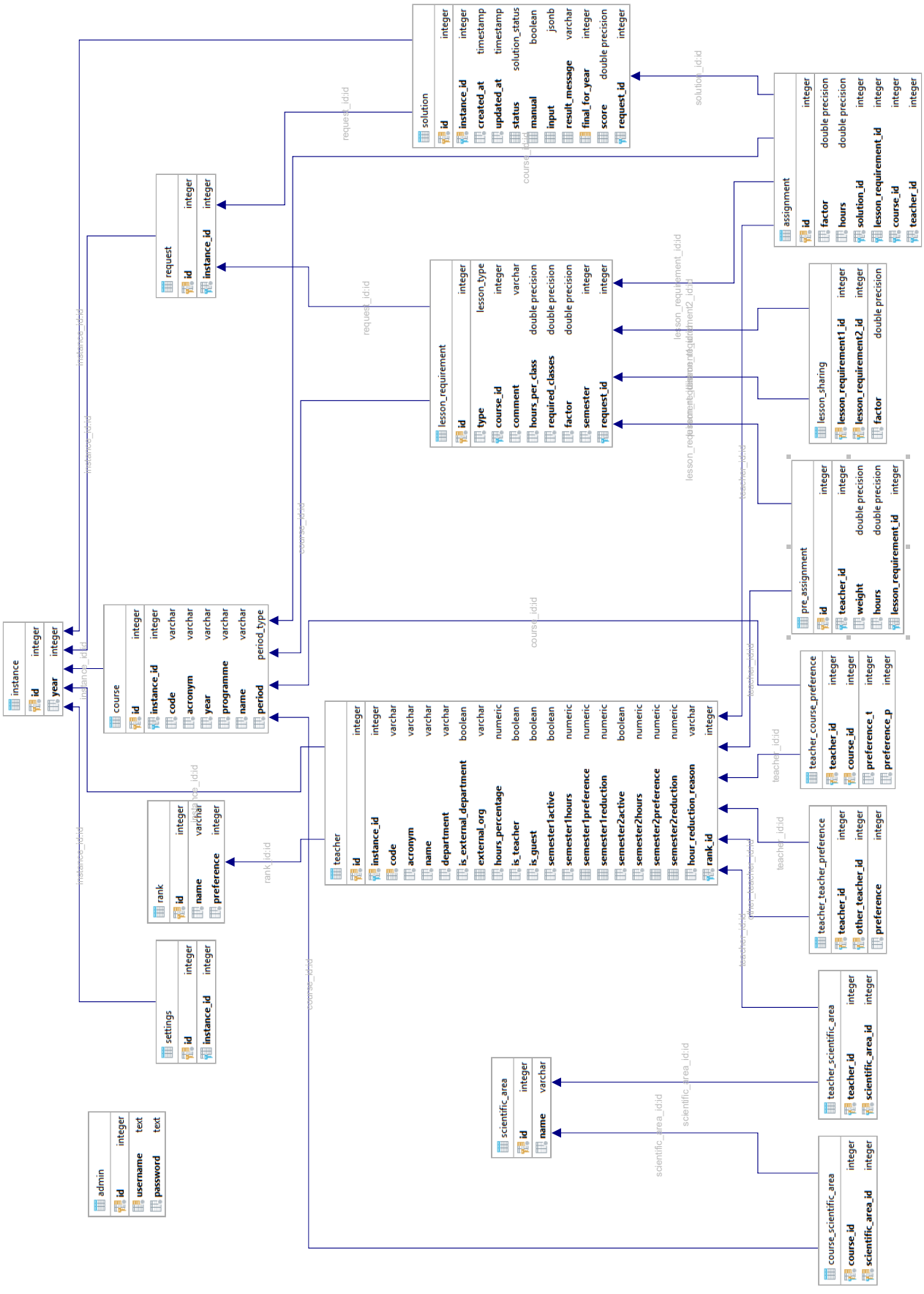

Figure 5.3: Database Diagram

- **Solution**: Represents a solution. There are two types of solutions. The ones imported from the data from SIGARRA and the ones generated by the system.
- **ScientificArea**: Represents a scientific area which can be used both for courses and teachers.
- **CourseScientificArea**: Association between a course and a scientific area.
- **TeacherScientificArea**: Association between a teacher and a scientific area.
- **LessonRequirement**: Represents a requirement for a type of lessons for a Course. Each requirement has a number of hours per class. For example, a class might take 2 hours per week. In addition, it has the number of classes that are assigned to the department for those lessons. This number is sometimes not an integer, usually when the department is only responsible for half of the lessons of a class. Multiple lesson requirements for a course can be used not only for different lesson types but also when part of the lessons are to be taught by a different teacher.
- **LessonSharing**: Represents the sharing of lessons between different courses. This link has an associated factor.
- **Assignment**: Represents an assignment of a teacher to a certain number of hours and corresponds to a lesson requirement.
- **PreAssignment**: Represents a pre-assignment with a certain weight made by the administrator of a teacher to a course. This can also be seen as a special type of preference.
- **TeacherTeacherPreference**: Represents the preference of a teacher towards another one with an associated preference level.
- **TeacherCoursePreference**: Represents the preference of a teacher towards a course with an associated preference level.

5.4 Front-end

A manual for the usage of the front-end was written, in Portuguese, for both the user and administrator interfaces. This manual is available in this dissertation in Appendix A. It was written in Portuguese due to this being the target audience's language.

As previously mentioned, the front-end was developed with Vue.js, a JavaScript front-end framework, with vue-cli[6] being used as a tool for bootstraping the project. Typescript, a superset of JavaScript which is transpiled to JavaScript, was used in order to add static typings to the language and to therefore help reducing the amount of bugs which can be caused by the lack of this feature.

[6]Available at: https://github.com/vuejs/vue-cli

Amongst other libraries, Vuex was used for global state management, Vue Router for creating pages and to route between them and ElementUI for some user interface components. All of the routing and rendering is done on the front-end by the browser and the JavaScript which runs on it. Websites which follow this type of pattern are known as single page applications (SPA). The front-end communicates with the back-end via a REST API using JSON for getting and sending the required data. This type of websites allow the users to get instant feedback instead of waiting for pages to load, which provides a more pleasant experience.

For authentication, the users are redirected to UP's Shibboleth which after a successful login redirects back to the platform. The platform then stores the user authorisation in a signed cookie.

As a security measure, both front-end and back-end follow the Double Submit Cookie pattern[OWASP, 2018b], which is a technique for preventing XSRF attacks. This means that the back-end in addition to the cookie for authorisation uses one for an anti-XSRF token. The authorisation cookie has the HTTPOnly flag set, which makes it not readable through the browser's JavaScript. This helps avoid major problems in case of a XSS vulnerability. The cookie for the anti-XSRF token does not have this flag set and is therefore readable. When the SPA does a request to the API, it puts this token in a custom HTTP header and the back-end checks if the header matches with the cookie. This interaction is illustrated in Fig. 5.4.

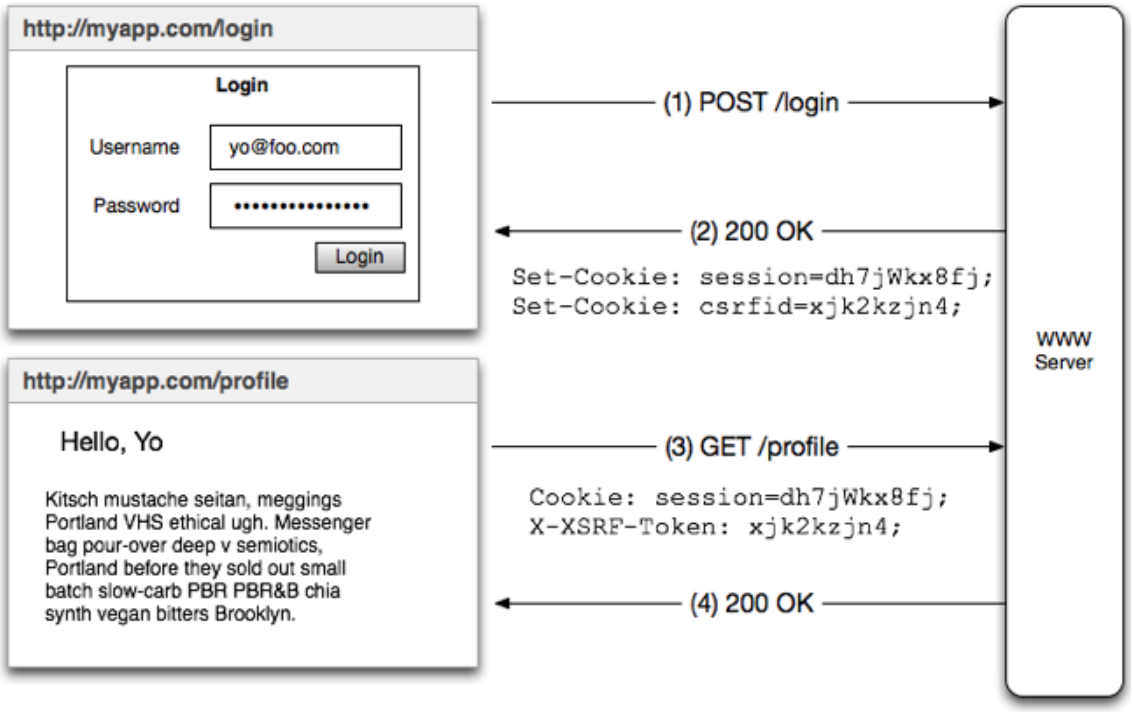

Figure 5.4: Double Submit Cookie Flow Diagram [Silverman, 2015]

In addition to the above, the front-end uses HTTPS with a TLS certificate, all the cookies are signed in order to avoid cookie tampering and are using the Secure flag, in order to avoid capture of the cookies by a man-in-the-middle attack, which could occur if the user connected to a insecure

network. The attacker could replace the page with a HTTP one, which would make the cookies readable to him.

As its possible to take away from the above, security was a major concern for this project.

In terms of functionality, the front-end allows teachers to login and input their preferences for courses, as shown in Fig. 5.5, preferences towards other teachers, similarly to what had been shown in the planning chapter, on Fig. 4.3, and workload preferences, as shown in Fig. 4.1.

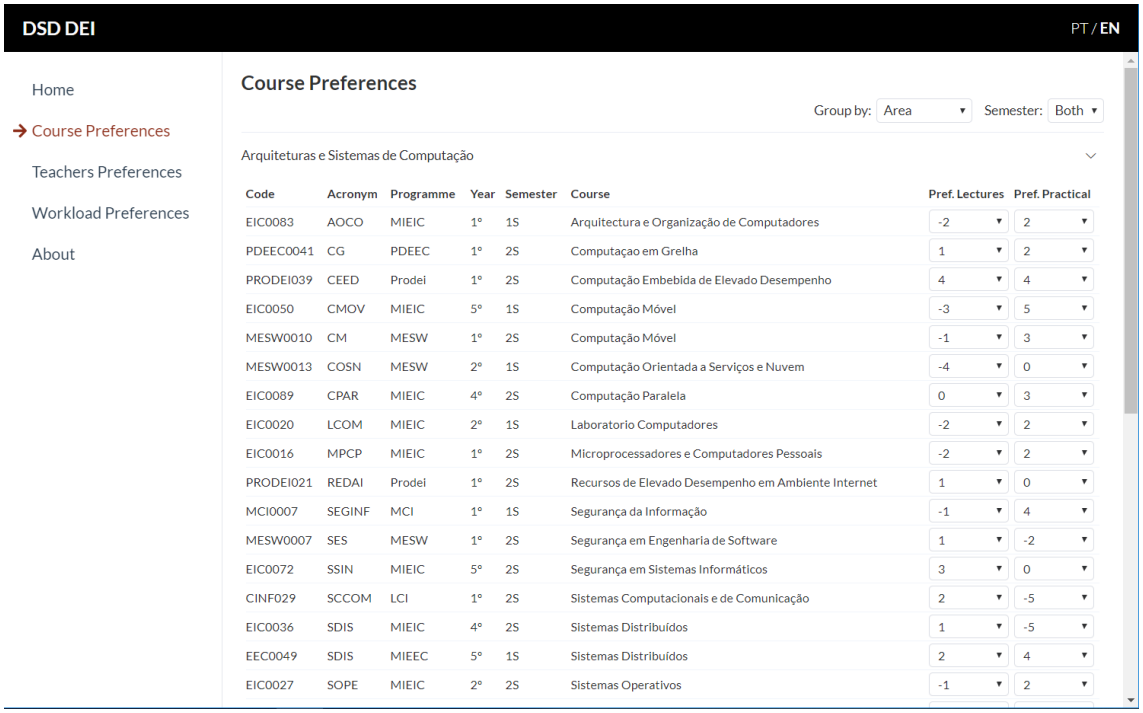

Figure 5.5: UI for course preferences selection

5.5 Back-end

As previously mentioned, the back-end was developed with Node.js, and like in the front-end, Typescript was also used. The Hapi.js library was used to ease the development of this component.

A process manager named pm2 was used in order to manage Node.js and to run multiple processes in cluster mode and load balance between them. This is useful as Node.js is single-threaded and this allows one to run it in a multithreaded manner.

In order to easily communicate with the PostgreSQL database, a ORM library was used with Objection.js being chosen, as it is a stable and mature choice. This allows the table rows to be easily mapped to objects and allows the creation of flexible and complex queries without appending strings in order to create SQL queries, which could lead to the unintended creation of SQL Injection vulnerabilities. An ORM also allows the user to query information which requires multiple

SQL queries in a easier way, with the ORM automatically merging the multiple responses into a single result.

All the inputs from the REST API are validated using a schema validation library, which helps avoid attacks where the attacker sends unexpected input to the API. For example, in cases where the system expects a positive number, the validation can prevent a client from sending a negative one.

In order to easily apply changes to the database, the used ORM has a migration system, which allows the creation of queries which are only run a single time if they have never been run before on that database. This information is stored on a specific database table. This library also expects one to create the reverse of each migration, in order to rollback the changes in case the update has problems and the developer wants to revert the system to the previous version.

The parsing of Excel files is done with the help of a library which exists for this purpose. The parsing is done in child processes of the server. This was done in order to avoid blocking Node.js' event loop[7] and to avoid any memory leaks which could have been caused by the parsing of these files. The child process is created and the files are sent to it as base64 encoded strings. The files are parsed and the results are inserted into the database. When this procedure is complete, a message is sent back to the parent process (the back-end server) in order to notify it that the process is complete. This process then kills the child in order to avoid using unnecessary memory resources.

SAML protocol, and therefore Shibboleth, consists of three different components: the User Agent (in this case the browser), the Service Provider (or SP, in this case this is the back-end server), and the Identity Provider (or IdP, which here is UP's Shibboleth server). The protocol flow is illustrated in Fig. 5.6.

The connection to UP's Shibboleth was done using a SAML library for Node.js which had to be patched in order to better work with Shibboleth and to get the required metadata from the login, which in this case is just the logged-in teacher's code. This code is then cross-referenced with the system's teachers database table. A public/private key pair was generated using RSA with 4096 bits and the system was registered with UP via a XML Metadata file which contains the Service Provider's public key. The Identity Provider's public key was also obtained from its XML Metadata file and inserted into the system. This way, both ends can sign their messages with the other end's public key, and data can the validated and checked for integrity when received with the respective private keys. This defends against man-in-the-middle attacks.

Admin accounts use a normal username and password login on the other hand. The passwords are hashed with the Argon2 algorithm[OWASP, 2018a], which is considered the state of the art in terms of password hashing algorithms, according to the OWASP Foundation[8].

RabbitMQ, a message broker which follows the AMQP protocol, was used in order to send tasks to the Solver and to receive updates from it. Since it allows clients to subscribe to channels,

[7] The event loop is what allows Node.js to perform non-blocking I/O operations (more information at: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/)

[8] The Open Web Application Security Project (OWASP) is a worldwide not-for-profit charitable organisation focused on improving the security of software (https://www.owasp.org)

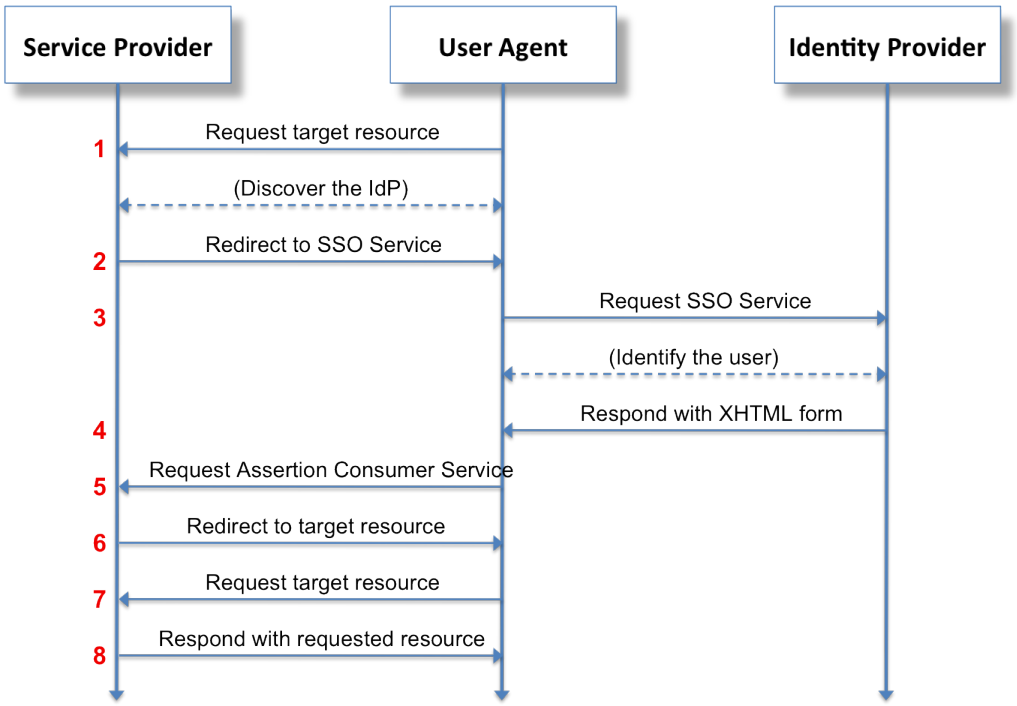

Figure 5.6: SAML Protocol Flow Diagram [Scavo, 2011]

the systems can be notified in real time, without doing any kind of polling. This is known as the Publish/Subscribe pattern.

The updates sent by the solver and mentioned on the previous paragraph, after being received by the back-end, are inserted into the database. Afterwards, these updates are broadcast to the front-end through websockets to any user which might be on the solver administration panel. The information is synchronised between back-end nodes using Redis. This means that even if the user is connected to a node different from the one which receives the update from the solver, the user still receives the update on the browser.

5.6 Message Broker

Figure 5.7 illustrates how RabbitMQ is being used in the system.

The "task_queue" is being used by the back-end server to send tasks to the solver. The solver sends status updates and results back to the server though the "results_queue". In order to command the solver to stop searching for a solution and to submit the best up to that moment, a stop task queue is used. There is a unique queue for each task. These last queues are created with the auto-delete property, which makes them be automatically deleted by RabbitMQ as soon as the consumer closes the connection, with the consumer being the solver in this case.

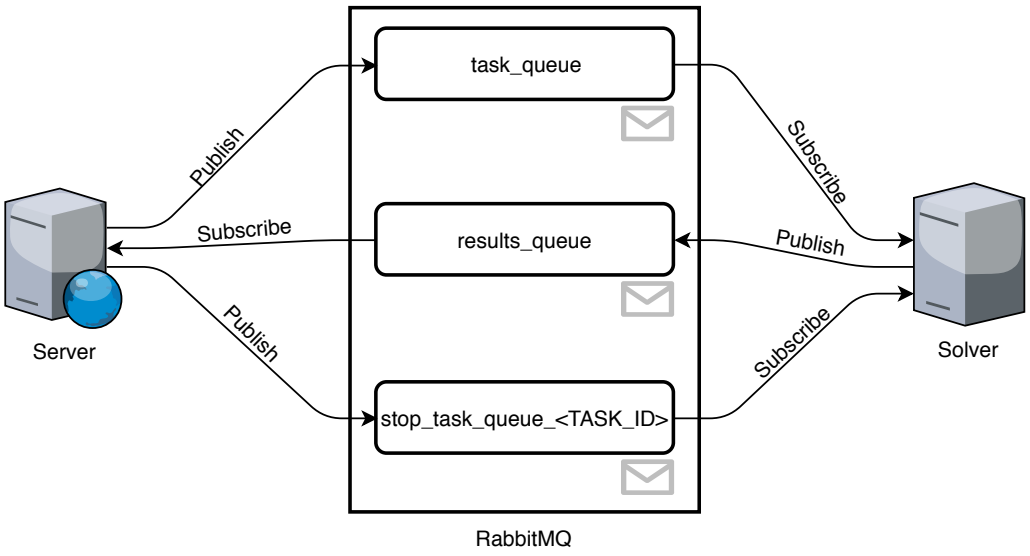

Figure 5.7: Message queues used by the system

5.7 Solver

The data received by the solver, from the message broker, consists of a JSON object with the following attributes:

- Task ID
- Current year
- List of teachers' course preferences
- List of teachers' preferences towards other teachers
- List of lesson requirements and related pre-assignments
- List of some information about each teacher: teacher ID in the database, how many hours are planned for each semester and information about external departments or schools.
- List of previous teacher-course assignments.

None of this information allows teachers to be easily identified, which allows their preferences to be kept confidential even for the people who might have access to the information being sent to and from the solver.

The information is then processed by the setup procedure of the solver into arrays, matrices or hash tables, in order to be efficiently used for the creation of the model to be used by CP Optimizer.

One extra dummy teacher is used in order to indicate the lessons to be taught by guest teachers, which are not assigned by the solver. This teacher has a negative preference value for all the courses.

The solver has several configuration parameters, which are:

- **Maximum hours factor**: Factor of the maximum number of hours a teacher can teach in relation to the expected number (e.g. 1.1). For example, with this value at 1.1, a teacher with a limit of 16 hours can teach up to 17 and a half hours.
- **Minimum hours factor**: Factor of the minimum number of hours a teacher must teach in relation to the expected number (e.g. 0.2) - this helps the solver reach better solutions more quickly.
- **Score course gain weight**: Factor of how much the teachers' course gains matter for the overall score. Here a gain is defined by the teachers' preferences with some extra artificial preferences added, such as the ones used to keep a teacher in a course for at least some years. These gains are also configurable and are also detailed in this section. This weight is multiplied by the average gain of the teachers in relation to the courses assigned to them.
- **Score average hours delta weight**: Negative factor of how much the average delta between assigned and expected workload matters.
- **Score maximum hours delta weight**: Negative factor of how much the maximum delta between assigned and expected workload matters.
- **Semester time balance weight**: Weight for how much importance should be given to the balancing of the hours between the two semesters, according to the preferences set by the teachers.
- **Score average courses per teacher weight**: Negative factor of how much the average number of courses per teacher matters.
- **Score maximum courses per teacher weight**: Negative factor of how much the maximum number of courses per teacher matters.
- **Score average new courses weight**: Negative factor of how much the average number of new courses per teacher matters.
- **Score maximum new courses weight**: Negative factor of how much the maximum number of new courses per teacher matters.
- **Score teacher preference weight**: Factor of how much the teachers' preferences towards other teachers matter for the overall score. This value is multiplied by the average satisfaction of the teachers in relation to the colleagues in the courses they are assigned to.
- **Desired hours per course**: The number of hours per course a teacher should have and which the solver should aim to obtain.
- **Minimum course years**: Number of years to try to keep a teacher in a course, by assigning an extra gain to their course preferences which are not negative (minimum course years gain increase).

- **Minimum course years gain increase**: Gain to add to the preferences of teachers to courses they had the previous year, for which they have not yet completed the minimum number of years and have not applied a negative preference to.
- **Keep course gain increase**: Gain to add to the preference of teachers to courses they had the previous year and have not assigned a negative preference to.
- **Pre-assignment gain multiplier**: Multiplier for the pre-assignment weights.
- **Guest teacher preference**: Preference for the dummy teacher, used to indicate a lesson will be taught by a guest. This should be a negative value in order to avoid the usage of guest teachers.

In order to convert the time from hours, with decimal places, to integers, these values are multiplied by 4 and rounded to the closest integer in order to represent quarters of an hour. In the end these are converted back to hours. This allows the solver to work only with time in integers, which makes it more efficient.

The preferences are normalised in order to avoid unbalanced choices making the system unfair. For example, one teacher putting a maximum preference in one course and the minimum preference on all the others should not be much different than the case where only that maximum preference is entered. In order to do this, all the positive and negative preferences are summed up into two different variables. The values are compared and the preferences on the side with a higher value, positive or negative, are multiplied by:

$$\frac{min(positivesSum, |negativesSum|)}{max(positivesSum, |negativesSum|)} \tag{5.1}$$

Considering a case where a teacher assigns a 3 to one course and -3 to 10 other ones, the -3 values would be converted to -0.3, making the preferences balanced, as the absolute of 10×-0.3 equals the the total of positive preferences.

In order to create a solution, an array of integer expressions is created representing the teacher assigned to each block. The array is created by going through each requirement and creating a block for each required lesson. Non integer values for the number of classes are dealt with by using the ceiling of the number and calculating the corresponding number of hours according to the following formula:

$$\frac{hours \times nClasses}{\lceil nClasses \rceil} \tag{5.2}$$

The course satisfaction for a block is calculated by taking the preferences for the course and type of lesson and using the *cp.element* method between all the teachers' preferences and the teacher assigned to the block. The result of this expression provides the satisfaction for whomever is assigned to the block. To each preference, extra points are added if the teacher taught that class the previous year (keep course gain increase parameter) and some others are added if the teacher has not taught it for a minimum number of years ("minimum course years gain increase" and "minimum course years" parameters). The preferences with these additional points are here

defined as "gain". The additional points are only added if the teacher has not given a negative preference to the course. The gain expressions for all the blocks are summed up and divided by the number of blocks, in order to get an average (*avgCourseGain*). These gains were added due to some policies of the department, which has the objective of avoiding a high number of changes from one year to the next.

The score penalty corresponding to the hours for each teacher (*hoursDelta*) is calculated according to the following expression, with *taughtHours* being the total number of hours taught by the teacher for the semester, and *S1taughtHours* and *S2taughtHours* the parts corresponding to the first and second semesters respectively.

$$\begin{aligned} &|taughtHours - desiredHours| + \\ &|S1taughtHours - desiredHoursS1| * semester\_time\_balance\_weight + \\ &|S2taughtHours - desiredHoursS2| * semester\_time\_balance\_weight \end{aligned} \tag{5.3}$$

The sum of the value divided by the number of teachers provided the average (*avgHoursDelta*) and the method *cp.max* provides the maximum (*maxHoursDelta*).

The number of courses per teacher is calculated by the following expression, where C is the number of courses and teacherCourses is a expression array which indicates if a teacher is assigned to at least a block of a given course.

$$\sum_{c=1}^{C} teacherCourses_c \tag{5.4}$$

The *teacherCourses* array is created by the following expression, with blockCourse being an integer array which returns the course for a given block:

$$\begin{aligned} teacherCourses[blockCourse[block]] = \\ (teacherCourses[blockCourse[block]] \\ + blockTeacher[block] = teacher) \geq 1 \end{aligned} \tag{5.5}$$

The value for each teacher is subtracted by the desired number of courses per teacher parameter and the absolute of the value is obtain through the *cp.abs* method. The maximum (maxCoursesPerTeacherDelta) is given by *cp.max* and the average (*avgCoursesPerTeacherDelta*) by dividing by the number of teachers.

The count of new courses per teacher is very similar to the *teacherCourses*, but with the difference of the expression only being applied to the courses which are new to the teacher, in comparison with the previous year. The average and maximum are assigned to the *avgTeacherNewCoursesCount* and *maxTeacherNewCoursesCount* variables, respectively.

A matrix is created for obtaining the preference between two given teachers. The value of each cell is the sum of the preference from teacher A to B and from B to A. The average satisfaction

for the teachers in relation to their colleagues in the courses they are teaching (*avgCourseColleagueSatisfaction*) is the average of the values obtained by for each course getting the preference of the teachers assigned to it in relation to any other teachers that might also be teaching that same course.

In order to help the system find a solution more quickly, the warm start feature from CPLEX CP Optimizer was used. It allows the model developer to create a solution that is used as a starting point for the algorithm. This works even if the solution is not feasible, as CPLEX is able to repair it. In order to do this, the previous year assignments as well as the requested pre-assignments are used in order to fill the blocks, by going through each and assigning the teachers to available blocks for the course, according to the requested number of hours.

Finally, taking into account the variables and weights explained above, the solver works by maximising the following score, with some of the components being positive and the others negative:

$$\begin{aligned} score = {} & avgCourseGain \times score\_course\_gain\_weight + \\ & avgHoursDelta \times score\_avg\_hours\_delta\_weight + \\ & maxHoursDelta \times score\_max\_hours\_delta\_weight + \\ & maxCoursesPerTeacherDelta \times score\_max\_courses\_per\_teacher\_weight + \\ & avgCoursesPerTeacherDelta \times score\_avg\_courses\_per\_teacher\_weight + \\ & avgTeacherNewCoursesCount \times score\_avg\_new\_courses\_weight + \\ & maxTeacherNewCoursesCount \times score\_max\_new\_courses\_weight + \\ & avgCourseColleagueSatisfaction \times score\_teacher\_pref\_weight \end{aligned} \tag{5.6}$$

5.8 Deployment

The system was deployed on two virtual machines.

The first one is running everything except the solver. In this machine, Docker, Docker Compose and Docker Machine were installed. Docker Machine allows the developer to easily connect to it through the Docker Engine and makes it easy to deploy the services there using Docker Compose.

In the second machine, Oracle Java SE Runtime Environment and IBM ILOG CPLEX 12.8.0 were installed. This machine runs the solver which communicates with the message broker present in the other one. The solver JAR was sent to it through SSH.

5.9 Summary and Conclusions

This section went into detail about the work which was developed for the project. A global overview of the architecture was made and the most relevant aspects of the main components

of the implementation, front-end, back-end, database, message broker and solver, were explained in more detail.

Several aspects were taken into consideration during the development of the project, namely security, performance, availability, scalability and maintainability. Security is a important factor as a vulnerability could lead to unauthorised access which would entail data privacy problems as well as possible tampering with the teachers' preferences. Performance, availability and scalability are important as well as the system should continue working even if the quantity of data and concurrent users increases. Finally, a maintainable system allows it to evolve and to continue being developed into the future.

Chapter 6

Results

In order to verify if the system works according to what is expected from it, some experiments and data analysis had to be done. This analysis includes mostly statistic measurements such as averages, minimums and maximums. In this chapter this process is explored and explained. In addition, some of the obtained results are shown.

6.1 Test Process Overview

The test process started by taking the files provided by SIGARRA and the programme directors and scientific area coordinators. These entities are illustrated in Fig. 5.1. This information includes teachers, roles, planned absences, courses and previous years assignments.

After having those files ready, some tests were done by assigning random preferences to teachers and testing the solver against that.

Afterwards, the data from the scientific areas coordinators had to also be modified in order to be used. These files had many inconsistencies. The Excel file for creating the assignments and the one for the requests for courses had inconsistencies between them and the official data from SIGARRA. These included incorrect teachers' initials and courses' codes. Data had to be manually verified and fixed. Furthermore, information from different Excel files had to be cross-referenced and merged. For this purpose, the Query Storm plugin for Excel was used. This plugin allows the user to run SQL queries on Excel tables, which really helped with this process, as it provided the ability to merge different tables with SQL.

Finally, the initial files were inserted into the system, the teachers were asked to fill their preferences, and afterwards, the course requests from the scientific area coordinators were inserted.

From the 37 permanent teachers, only 9 filled in any course preferences, with only 3 filling preferences for other teachers. However, the system should be able to work properly even if few teachers fill in their preferences, as it is assumed they are satisfied with they current ones.

The input data includes 172 teachers, from which 76 are from the department and not external. The solver fully assigns these 76 teachers to courses, while the rest are only assigned based on

the pre-assignments. There are 224 courses and 239 lesson requirements. Based on the number of required lessons, these result in 440 lesson blocks for the solver.

As stated previously, this data set is considered quite large when compared to the others mentioned in the literature.

The experiments were run on a virtual machine with 8 threads and 24GB of RAM assigned to it, running on a Intel(R) Xeon(R) E5-2620 v3 CPU clocked at 2.40GHz. The latest version of CPLEX at the time of writing, 12.8, was used for the experiments.

6.2 Result Analysis

The solver was configured with the following parameters, which were arrived at by the execution of previous experiments and changing the weights according to what was observed from them:

$maximum\_hours\_factor = 1.1$
$minimum\_hours\_factor = 0.7$
$score\_course\_gain\_weight = 5$
$score\_avg\_hours\_delta\_weight = -0.1$
$score\_max\_hours\_delta\_weight = -0.05$
$semester\_time\_balance\_weight = 0.2$
$score\_avg\_courses\_per\_teacher\_weight = -1$
$score\_max\_courses\_per\_teacher\_weight = -1$
$score\_avg\_new\_courses\_weight = -1$
$score\_max\_new\_courses\_weight = -10$
$score\_teacher\_pref\_weight = 1$
$desired\_hours\_per\_course = 6$
$min\_course\_time\_years = 3$
$min\_course\_time\_gain\_increase = 3$
$keep\_course\_gain\_increase = 1$
$pre\_assignment\_gain\_multiplier = 3$
$guest\_teacher\_gain = -6$

It was then run for 24 hours. During this time, a optimal solution was not able to be obtained and the objective function resulted in a score of around -150.97.

Table 6.1 shows the results of some of the assignments created by the solver.

Afterwards, the teachers preferences expression was removed from the objective function in order to simplify the model, as this expression was what lead to highest increase in complexity. After removing this expression, and running the solver again for 24 hours, a score of around -22.2890 was obtained, as shown in Fig. 6.1, with a upper bound reported by CPLEX CP Optimizer of 5.1556. The progress of the score during those 24 hours is shown in Fig. 6.2. The final solution has obtained at around the 18h30 mark.

This score of -22.2890 is composed of the sub-score components shown in Table 6.2.

teacher_id	course_id	factor	hours	type	hours_per_class	semester
101	222	1	2	OT	2	2
152	221	1	1	TP	1	2
98	220	1	2	TP	2	2
83	212	1	2	L	2	1
12	210	1	1	S	1	1
51	209	1	1	S	1	1
133	208	1	1	T	1	1
119	207	1	0.75	T	0.75	1

Table 6.1: Examples of generated assignments

The average course gain had a value of around 7.36 which means teachers' preferences and artificial gains were followed.

Values of -0.24 and -0.65 for the average hours delta and max hours delta are positive results. These correspond to an average of 0.42 hours of overtime in relation to the target hours for each teacher and deltas ranging from -0.5 and 1.5.

The teachers who filled in any preferences had an average course satisfaction of around 2.103 out of 3, which means that for the most part their preferences were respected. The value for each of these teachers is represented in Table 6.3.

The courses per teacher delta values correspond to an average of 4.54 for the permanent teachers, which is close to the value of 5 obtained for the previous year's assignments.

Finally, the new courses per teacher had an average value of 0.53, which is better than the value of 0.75 obtained in the previous year. The maximum value was of a teacher with two new courses, which caused this score component to have a value of -20. This weight was set so high because it is important that a teacher should not get many new courses in a year.

The results matrix, where the generated assignments can be manually edited, is shown in Fig. 6.3. As previously mentioned, this interface allows the administrator to manually edit the automatically generated assignments in order to fix any potential undesired allocations made by the system and to fine-tune the final result.

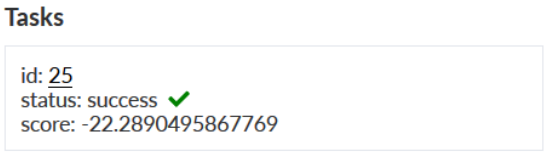

Figure 6.1: Result score

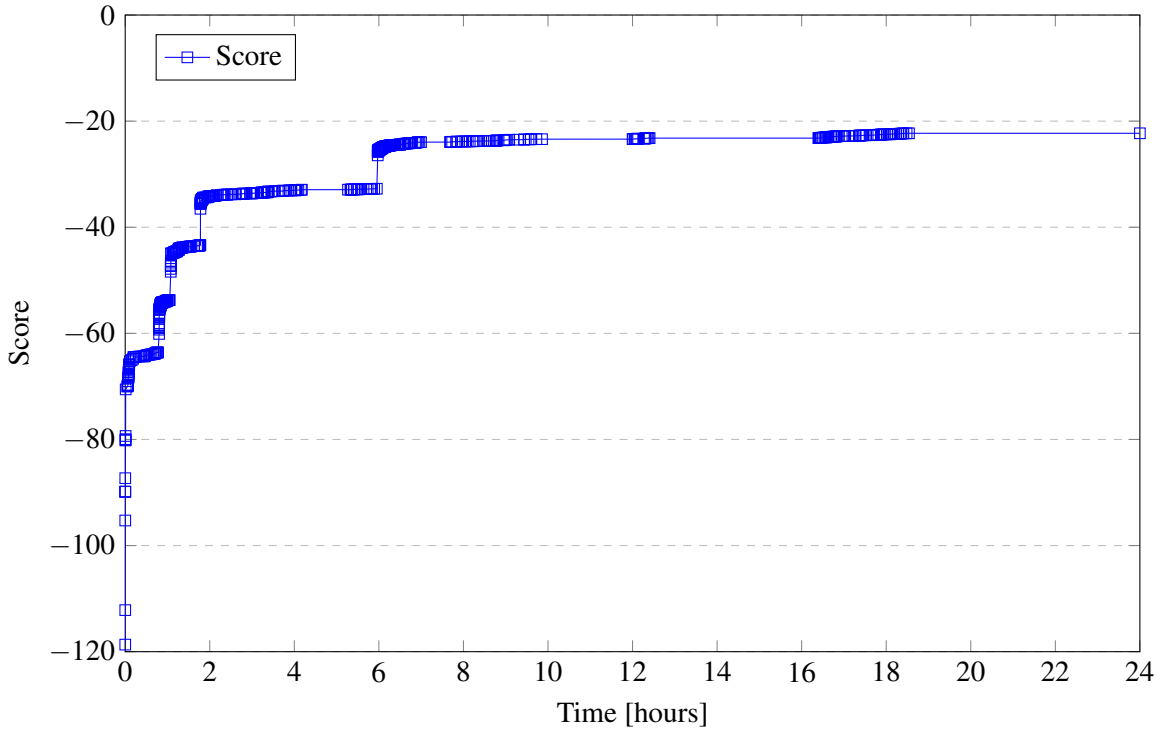

Figure 6.2: Score over time

component	score
$avgCourseGain \times weight$	7.360950413223159
$avgHoursDelta \times weight$	−0.2436046511627909
$maxHoursDelta \times weight$	−0.65
$maxCoursesPerTeacherDelta \times weight$	−9.0
$avgCoursesPerTeacherDelta \times weight$	−1.7034883720930245
$avgTeacherNewCoursesCount \times weight$	−0.5348837209302332
$maxTeacherNewCoursesCount \times weight$	−20.0

Table 6.2: Score components

teacher_id	average course preference
11	3
49	3
66	1.8235294117647058
74	3
100	2.342857142857143
101	1.0588235294117647
110	2.2941176470588234
129	0.7058823529411765
188	1.7058823529411764

Table 6.3: Average course satisfaction per teacher

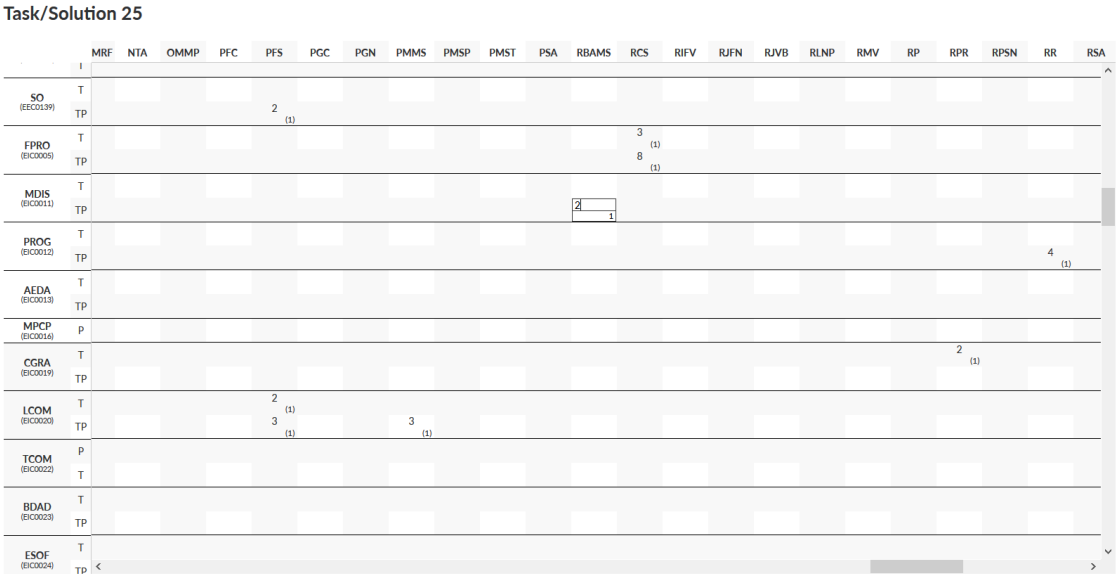

Figure 6.3: Editable results matrix

Chapter 7

Conclusions and Future Work

This chapter summarises all the work developed herein, as well as the conclusions that have been made, including limitations and possible future work.

7.1 Conclusions

Every year the Department of Informatics Engineering (DEI) of the Faculty of Engineering of the University of Porto (FEUP) has to assign each teacher to the classes which are to be taught. This is currently done manually, mostly by just slightly changing the previous year assignments. The current process does not take into account the teachers' preferences which in turn lowers their satisfaction.

A system for solving this problem, composed of three steps, was proposed and developed. These steps consist of the collection of the teachers' preferences, the automatic generation the assignments between teachers and courses, and finally, the manual editing of the generated solution by the administrator of the system.

This developed platform includes a back-end server, which handles all the requests made by the front-end, including the parsing of the input files, front-end web interfaces for both teachers and administrators, built as a single page application, a database where all the data resides on, a solver, which uses constraint programming for the assignment of teachers to lesson blocks, as well as a message broker and a in-memory database for some communication and synchronisation between components.

The built solver produced interesting results, specially taking into account the fact that the used input files from the programme and scientific area directors were not the respective final versions which have more pre-assignments which would help arrive at a better solution.

7.2 Future Work

Since the solver is not able to prove optimality, at least in 24 hours, future work should aim to solve this issue. Even though the score graph seems to converge after some hours, it would be

positive if optimality could be proven.

This might be possibly done by improving the model, with the usage of different features of CP Optimizer which might be able to help increase its efficiency. For example, the implementation of a constraint propagator, as the one described by Richter et al. [2008], could help prune the search space in order to arrive at a acceptable solution more quickly.

If the run time was to be reduced to a few minutes instead of many hours, the system could be used in a more iterative way where the administrator could make changes and run the solver multiple times. As it stands currently, one is expected to provide the input files and run the solver a single time and make any necessary changes manually afterwards. With a lower run time, the changes could be made to the input files instead by adding new constraints in the form of pre-assignments, with the solver being run again with the updated files until a satisfactory solution was obtained.

Further work can also be done in order to fine-tune the solver to arrive at better solutions. This can be done by adding constraints which might help restrict the search space and by changing the weights of the objective function in a way which will better guide the search.

In addition to the above, improvements can be made to the web app, in terms of functionally and usability and the integration with SIGARRA could be automated through a webservice which would have to be provided by the University.

Furthermore, the imported data could include the feedback provided by the students in relation to the teachers' performance in the respective courses. This data could also be used in addition to the teachers' preferences.

The system could also be improved to better handle the hiring of teaching assistants, by using a list of possibly available ones and suggesting assignments to courses for which there are not enough available teachers, taking into consideration their preferences for scientific areas or even specific courses.

In order to help improve the quality of the solution, a what-if analysis system could be developed, which would help the administrator test certain scenarios, such as testing how the assignments would be affect by changing certain parameters.

Finally, further metrics could be added, such as balancing the number of theoretical classes being taught and preferences in relation to programmes.

References

Salem M Al-Yakoob and Hanif D Sherali. Mathematical programming models and algorithms for a class–faculty assignment problem. *European Journal of Operational Research*, 173(2): 488–507, sep 2006. doi: 10.1016/j.ejor.2005.01.052.

Gary M. Andrew and Robert Collins. Matching Faculty to Courses. *College and University*, 46 (2):83–89, 1971.

Sigal Asaf, Haggai Eran, Yossi Richter, Daniel P Connors, Donna L Gresh, Julio Ortega, and Michael J Mcinnis. Applying Constraint Programming to Identification and Assignment of Service Professionals. In *Principles and Practice of Constraint Programming – CP 2010*, pages 24–37. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-15396-9_5.

Pasquale Avella and Igor Vasil'Ev. A Computational Study of a Cutting Plane Algorithm for University Course Timetabling. *Journal of Scheduling*, 8(6):497–514, dec 2005. doi: 10.1007/s10951-005-4780-1.

Jon A. Breslaw. A linear programming solution to the faculty assignment problem. *Socio-Economic Planning Sciences*, 10(6):227–230, jan 1976. doi: 10.1016/0038-0121(76)90008-2.

Marco P. Carrasco and Margarida V. Pato. A Multiobjective Genetic Algorithm for the Class/Teacher Timetabling Problem. In *Practice and Theory of Automated Timetabling III*, pages 3–17, 2001. doi: 10.1007/3-540-44629-X_1.

M.P. Carrasco and M.V. Pato. A comparison of discrete and continuous neural network approaches to solve the class/teacher timetabling problem. *European Journal of Operational Research*, 153 (1):65–79, feb 2004. doi: 10.1016/S0377-2217(03)00099-7.

Michael W Carter and Gilbert Laporte. Recent developments in practical course timetabling. In *Practice and Theory of Automated Timetabling II*, pages 3–19. Springer, Berlin, Heidelberg, 1998. doi: 10.1007/BFb0055878.

DigitalOcean™ Inc. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems, 2014. Available online at: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. Accessed on 2018-01-22.

B Domenech and A Lusa. A MILP model for the teacher assignment problem considering teachers' preferences. *European Journal of Operational Research*, 249(3):1153–1160, mar 2016. doi: 10.1016/j.ejor.2015.08.057.

James S. Dyer and John M. Mulvey. An Integrated Optimization/Information System for Academic Departmental Planning. *Management Science*, 22(12):1332–1341, aug 1976. ISSN 0025-1909. doi: 10.1287/mnsc.22.12.1332.

REFERENCES

Aldy Gunawan and Kien Ming Ng. Solving the teacher assignment problem by two metaheuristics. *International Journal of Information and Management Sciences*, 22(1):73–86, 2011. doi: 10.6186/IJIMS.2011.22.1.5.

Aldy Gunawan, Kien Ming Ng, and Kim Leng Poh. Solving the Teacher Assignment-Course Scheduling Problem by a Hybrid Algorithm. *International Journal of Computer, Information, and System Science, and Engineering*, 1(2):136–141, 2007.

Aldy Gunawan, K. M. Ng, and H. L. Ong. A genetic algorithm for the teacher assignment problem for a University in Indonesia. *International Journal of Information and Management Sciences*, 19(1):1–16, 2008.

Hans Mittelmann. Mixed Integer Linear Programming Benchmark (MIPLIB2010), 2018. Available online at: http://plato.asu.edu/ftp/milpc.html. Accessed on 2018-01-31.

Ali Hmer and Malek Mouhoub. Teaching Assignment Problem Solver. In *Trends in Applied Intelligent Systems*, pages 298–307. Springer, 2010. doi: 10.1007/978-3-642-13025-0_32.

Tim H Hultberg and Domingos M Cardoso. The teacher assignment problem: A special case of the fixed charge transportation problem. *European Journal of Operational Research*, 101(3): 463–473, sep 1997. doi: 10.1016/S0377-2217(96)00082-3.

IBM. IBM ILOG CPLEX Optimization Studio Getting Started with CPLEX, 2017. Available online at: https://www.ibm.com/support/knowledgecenter/SSSA5P{_}12.8.0/ilog.odms.studio.help/pdf/gscplex.pdf. Accessed on 2018-02-05.

IBM. IBM ILOG CPLEX Optimizer performance benchmarks | IBM Analytics, 2018. Available online at: https://www.ibm.com/analytics/data-science/prescriptive-analytics/optimization-software. Accessed on 2018-02-04.

Philippe Laborie. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 148–162, 2009. doi: 10.1007/978-3-642-01929-6_12.

Philippe Laborie and Bilal Messaoudi. New Results for the GEO-CAPE Observation Scheduling Problem. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pages 382–390, 2017.

Kai Lei, Yining Ma, and Zhi Tan. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python and Node.js. In *Proceedings of the 17th IEEE International Conference on Computational Science and Engineering, CSE 2014, Jointly with 13th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2014, 13th International Symposium on Pervasive Syst*, pages 661–668. IEEE, dec 2015. ISBN 9781479979813. doi: 10.1109/CSE.2014.142.

Richard H. McClure and Charles E. Wells. A Mathematical Programming Model For Faculty Course Assignments. *Decision Sciences*, 15(3):409–420, 1984.

Microsoft. The MVVM Pattern. Microsoft patterns & practices. Proven practices for predictable results., 2012. Available online at: https://msdn.microsoft.com/en-us/library/hh848246.aspx. Accessed on 2018-01-22.

REFERENCES

José Joaquim Moreira and Luís Paulo Reis. Multi-Agent System for Teaching Service Distribution with Coalition Formation. In *Advances in Information Systems and Technologies*, pages 599–609, 2013. doi: 10.1007/978-3-642-36981-0_55.

Tomas Eric Nordlander. Constraint Programming (CP), 2009. Available online at: https://www.sintef.no/globalassets/project/evitameeting/tomas-nordlander---cp-presentation---evita-2009---final-.pdf. Accessed on 2018-02-05.

OWASP. Password Storage Cheat Sheet, 2018a. Available online at: https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet#Leverage_an_adaptive_one-way_function. Accessed on 2018-06-20.

OWASP. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet, 2018b. Available online at: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#Double_Submit_Cookie. Accessed on 2018-06-20.

Fariborz Y. Partovi and Bay Arinze. A Knowledge Based Approach to the Faculty-course Assignment Problem. *Socio-Economic Planning Sciences*, 29(3):245–256, 1995.

Tianbao Qin, Yuquan Du, and Mei Sha. Evaluating the solution performance of IP and CP for berth allocation with time-varying water depth. *Transportation Research Part E: Logistics and Transportation Review*, 87:167–185, mar 2016. doi: 10.1016/j.tre.2016.01.007.

Luís Paulo Reis. *Coordenação em Sistemas Multi-Agente: Aplicações em Futebol Robótico e Gestão da Universidade*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2003.

Yossi Richter, Yehuda Naveh, Donna L Gresh, and Daniel P Connors. Optimatch: applying constraint programming to workforce management of highly skilled employees. *International Journal of Services Operations and Informatics*, 3(3/4):258–270, 2008. doi: 10.1504/IJSOI.2008.021338.

Tom Scavo. Security Assertion Markup Language (SAML) Diagram, 2011. Available online at: https://en.wikipedia.org/wiki/File:Saml2-browser-sso-redirect-post.png. Accessed on 2018-06-24.

Micah Silverman. The Problem with Securing Single Page Applications, 2015. Available online at: https://stormpath.com/blog/secure-single-page-app-problem. Accessed on 2018-06-24.

StackOverflow. Stack Overflow Developer Survey 2018 - Most Popular Databases, 2018. Available online at: https://insights.stackoverflow.com/survey/2018/#technology-databases. Accessed on 2018-07-03.

Phantipa Thipwiwatpotjana. Course assignment problem with interval requested workload. In *Proceedings of the 2014 IEEE Conference on Norbert Wiener in the 21st Century (21CW)*, pages 1–5. IEEE, 2014. doi: 10.1109/NORBERT.2014.6893875.

P.I. Tillett. An operations research approach to the assignment of teachers to courses. *Socio-Economic Planning Sciences*, 9(3-4):101–104, jun 1975. doi: 10.1016/0038-0121(75)90018-X.

REFERENCES

Lorraine Trilling, Alain Guinet, and Dominiue Le Magny. Nurse Scheduling Using Integer Linear Programming And Constraint Programming. *IFAC Proceedings Volumes*, 39(3):671–676, 2006. doi: 10.3182/20060517-3-FR-2903.00340.

Vue.js. Comparison with Other Frameworks - Vue.js, 2016. Available online at: https://vuejs.org/v2/guide/comparison.html. Accessed on 2018-01-22.

Yen-Zen Wang. An application of genetic algorithm methods for teacher assignment problems. *Expert Systems with Applications*, 22(4):295–302, 2002. doi: 10.1016/S0957-4174(02)00017-9.

Ian David Wilson, Ross Davies, and Nigel Stanton. A Genetic Algorithm based Solution to the Teaching Assignment Problem. *International Journal of Computer Applications*, 81(19), 2013.

Apêndice A

Manual de Utilizador

A.1 Manual do Professor

O utilizador, ao aceder ao *URL* `https://dsddei.fe.up.pt`, deverá encontrar o ecrã inicial, correspondente à Fig. A.1. Neste encontra-se um botão para ir para a página de *login*. Para além disto, no canto superior direito, encontram-se dois botões, com texto "PT"e "EN"que permitem trocar a interface para português e inglês, respetivamente.

Figura A.1: Página inicial

Após pressionar o botão mencionado acima, o utilizador é redirecionado para a página de autenticação federada da Universidade do Porto, ilustrado na Fig. A.2. Deverá introduzir as suas credenciais da Universidade para entrar no sistema.

Figura A.2: Autenticação federada da Universidade do Porto

No caso de o utilizador ter permissão para aceder ao sistema, isto é, estar na base de dados de professores desta plataforma, deverá encontrar o ecrã correspondente à Figura A.3, onde encontra uma barra lateral para navegar pelas várias vistas da interface.

Figura A.3: Página inicial após login

Mudando a página para inglês, através do botão no canto superior direito, referido anteriormente, é possível observar a mudança do texto de português para inglês. Isto é ilustrado na Fig. A.4.

Figura A.4: Alteração da interface para inglês

Ao aceder à página de preferências de Unidades Curriculares, é possível observar uma lista das áreas cientificas disponíveis, como é possível observar na Fig. A.5.

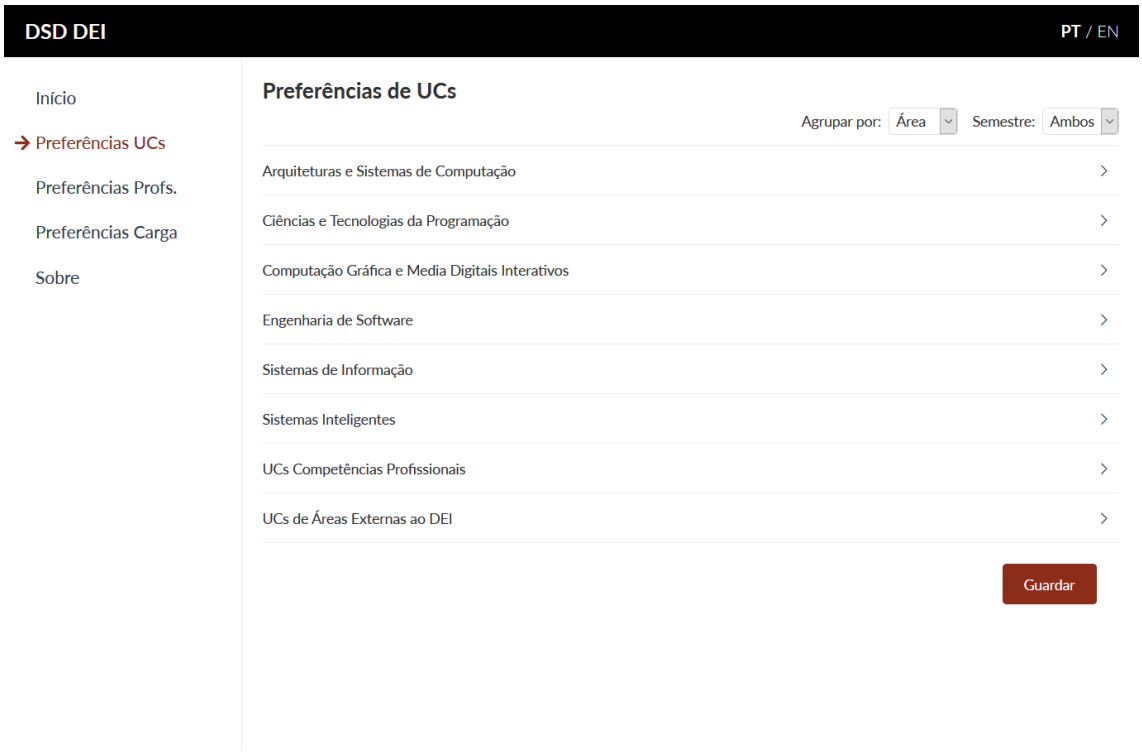

Figura A.5: Página de preferências de unidades curriculares

Após clicar numa das áreas, esta expande e é possível aceder às várias unidades curriculares da área, como é possível visualizar na Fig. A.6.

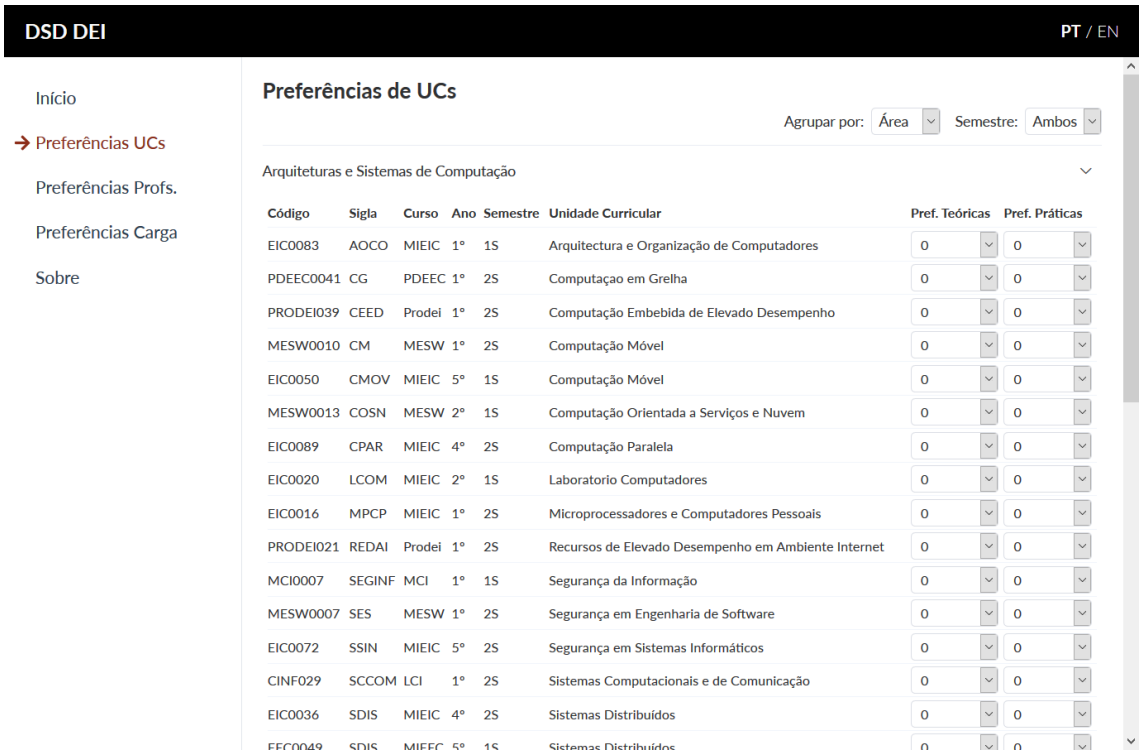

Figura A.6: Visualização de preferências de unidades curriculares

É também possível alterar o agrupamento das unidades curriculares. Para além de agrupadas por área, como foi observado anteriormente, estas podem ser agrupadas por curso. Este tipo de agrupamento encontra-se representado pela Fig. A.7.

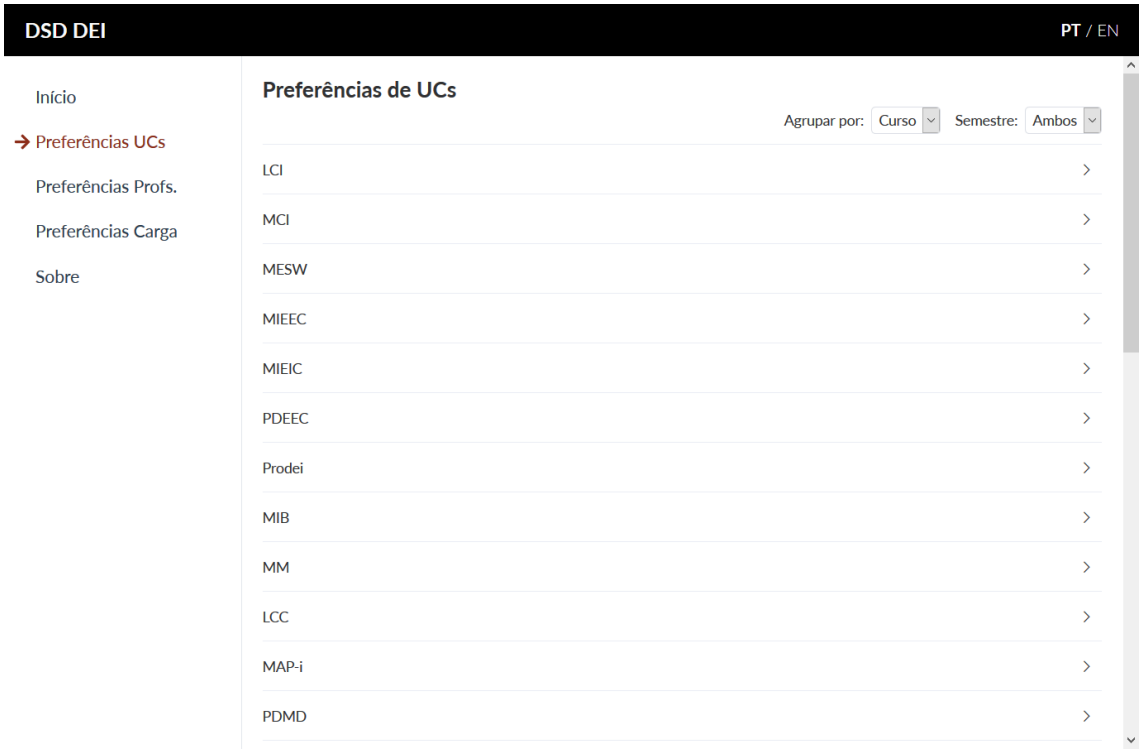

Figura A.7: Unidades curriculares agrupadas por curso

Para além disto, pode-se filtrar por semestre, como é visível na Fig. A.8.

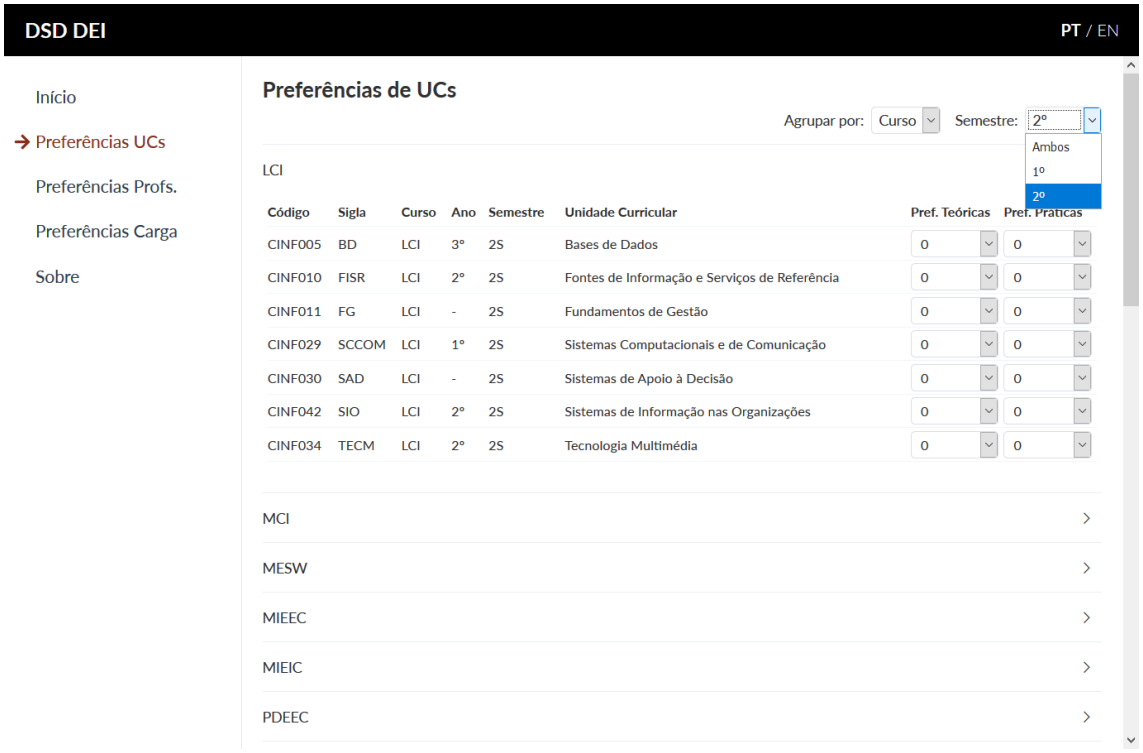

Figura A.8: Filtragem de unidades curriculares por semestre

Na Fig. A.9 é possível visualizar a definição de preferências, que pode ser feito tanto para aulas teóricas como práticas. Após a realização de alterações, o utilizador deve pressionar o botão de "Gravar"que se encontra no final da página.

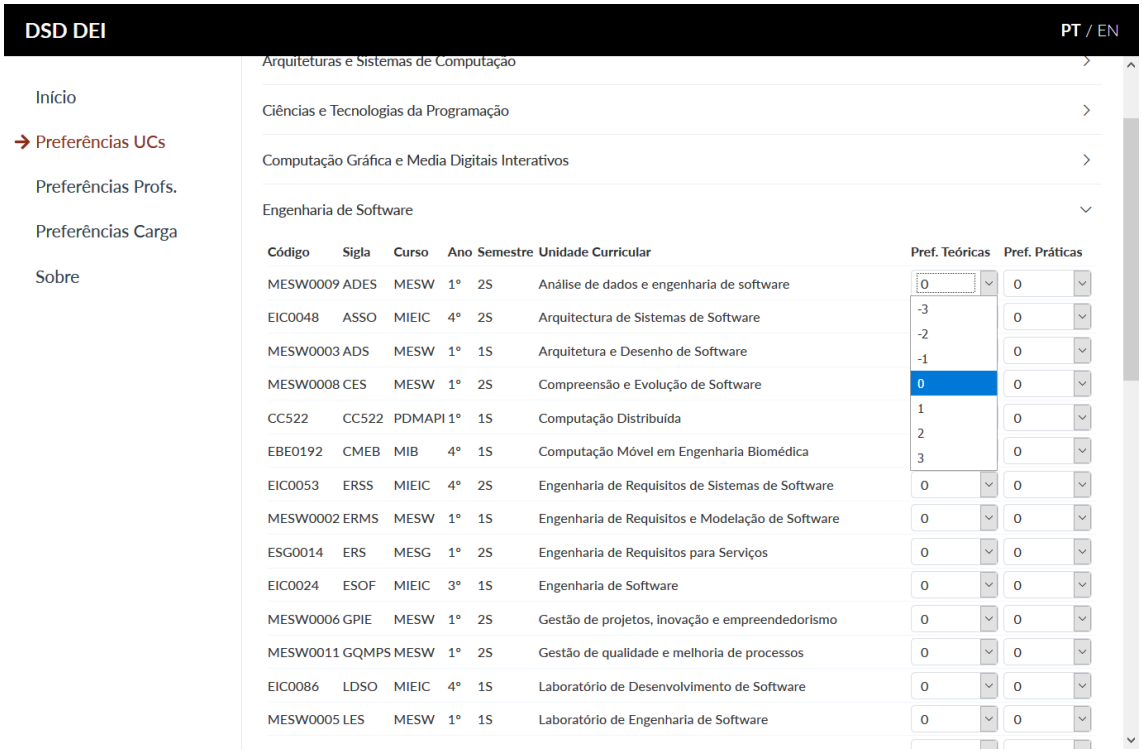

Figura A.9: Definição de preferências de unidades curriculares

Clicando no botão para Preferências de Professores, na barra lateral esquerda, o utilizador é redirecionado para a página para definição de preferências em relação a outros professores. Nesta página os utilizadores encontram-se organizados em dois grupos diferentes: professores de quadro e professores convidados. Esta página encontra-se ilustrada na Fig. A.10.

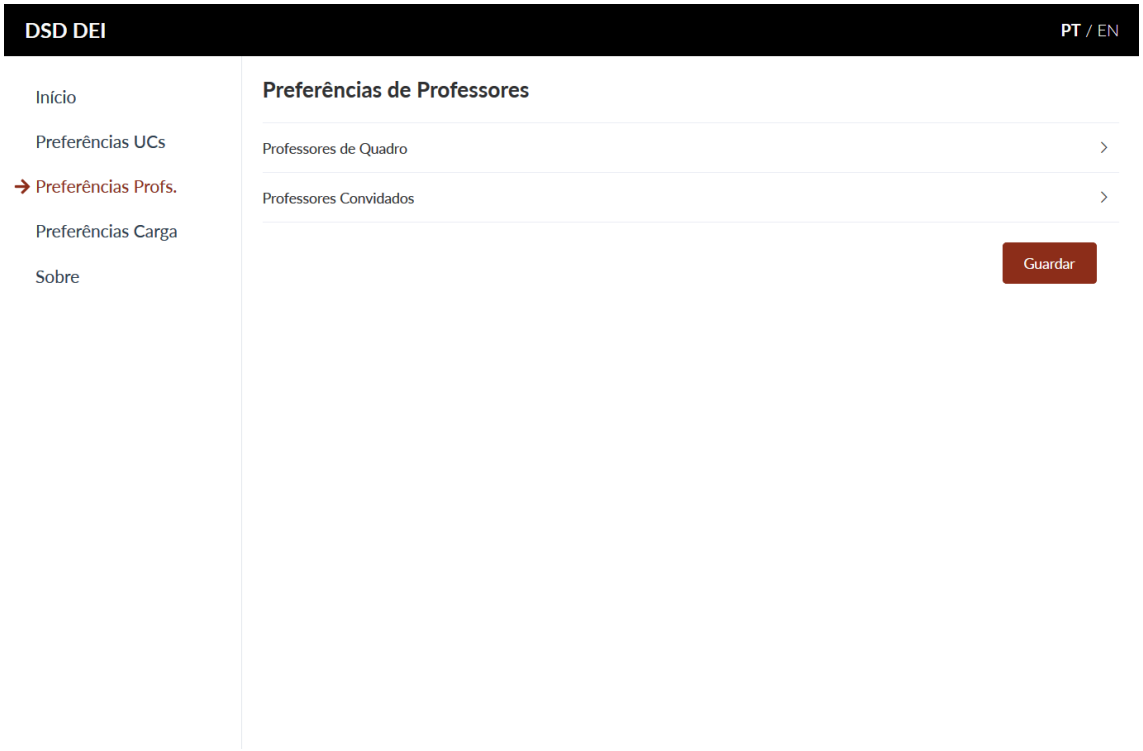

Figura A.10: Página de preferências de professores

À semelhança das preferências de unidades curriculares, os utilizadores podem definir preferências, como mostrado na Fig. A.11.

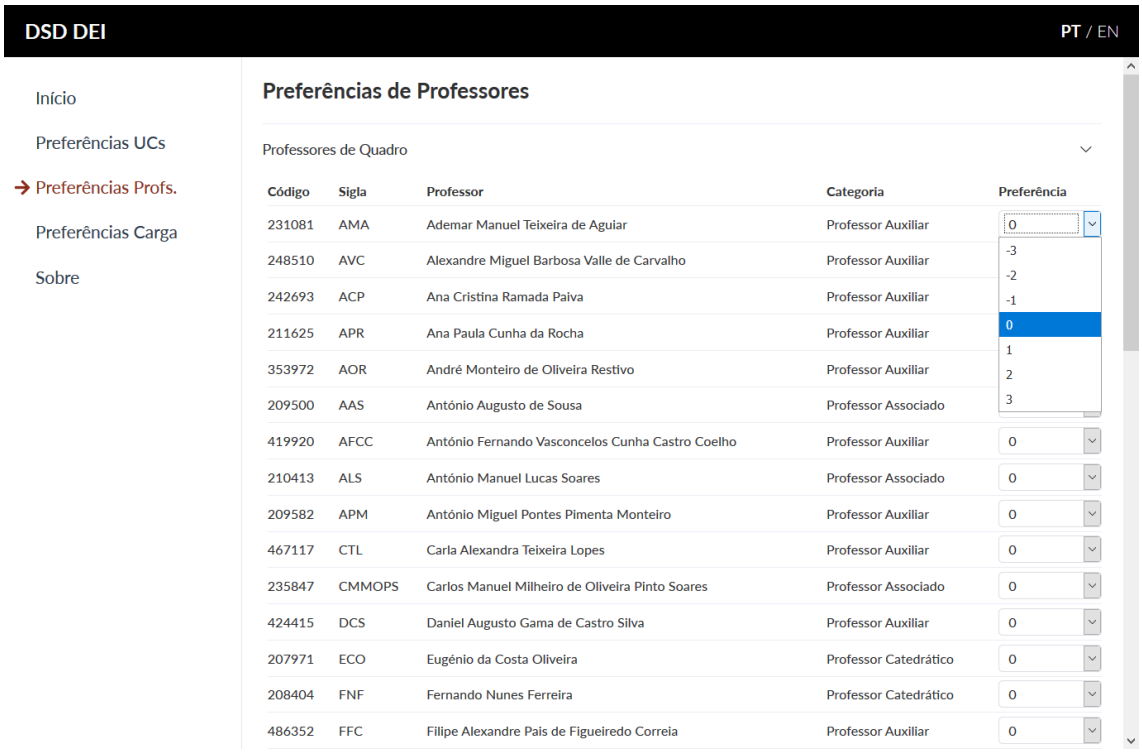

Figura A.11: Alteração de preferências em relação a um professor

Finalmente, na página de Preferências de Carga de Trabalho, é possível definir as preferências de horas semanais em cada um dos dois semestres. Este números podem ser definidos de três formas diferentes: arrastando o indicador que se encontra na barra; alterando diretamente os valores que se encontram indicados por "Primeiro Semestre"e "Segundo Semestre"; e utilizando os botões "+"e -"ao lado destes mesmos valores. Esta interface pode ser visualizada na Fig. A.12.

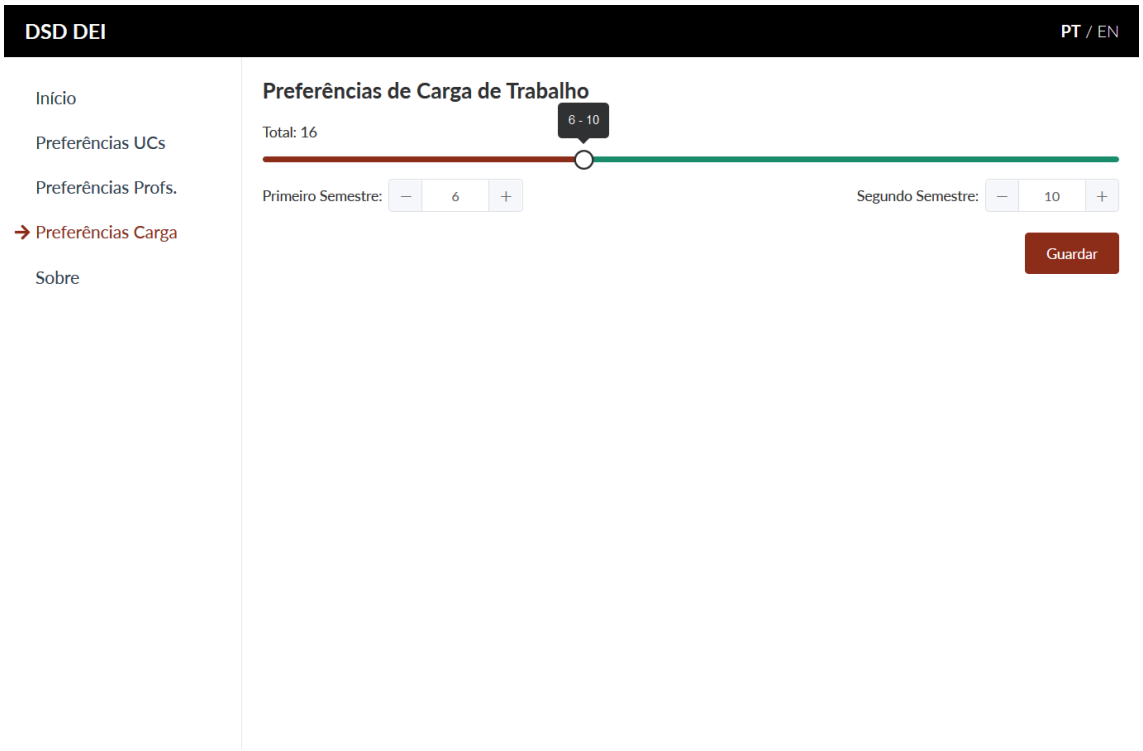

Figura A.12: Alteração de preferências de carga de trabalho

A.2 Manual de Administrador

O utilizador, ao aceder ao *URL* `https://dsddei.fe.up.pt/admin`, deverá encontrar o ecrã de login de administração, correspondente à Fig. A.13. Deverá introduzir o *username* e *password* corretos para prosseguir.

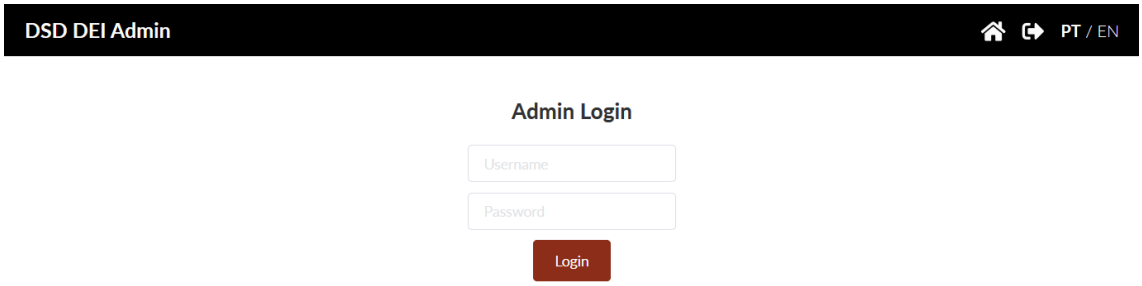

Figura A.13: Página de login para administração

Após *login* bem sucedido, surge o painel de administração. A navegação pode ser feita através da barra lateral esquerda, à semelhança da interface dos professores. No ecrã de início, presente na Fig. A.14, é possível ativar e desativar a interface dos professores, de modo a permitir bloquear o acesso dos mesmos à plataforma, durante o espaço de tempo em que não se realiza recolha de preferências.

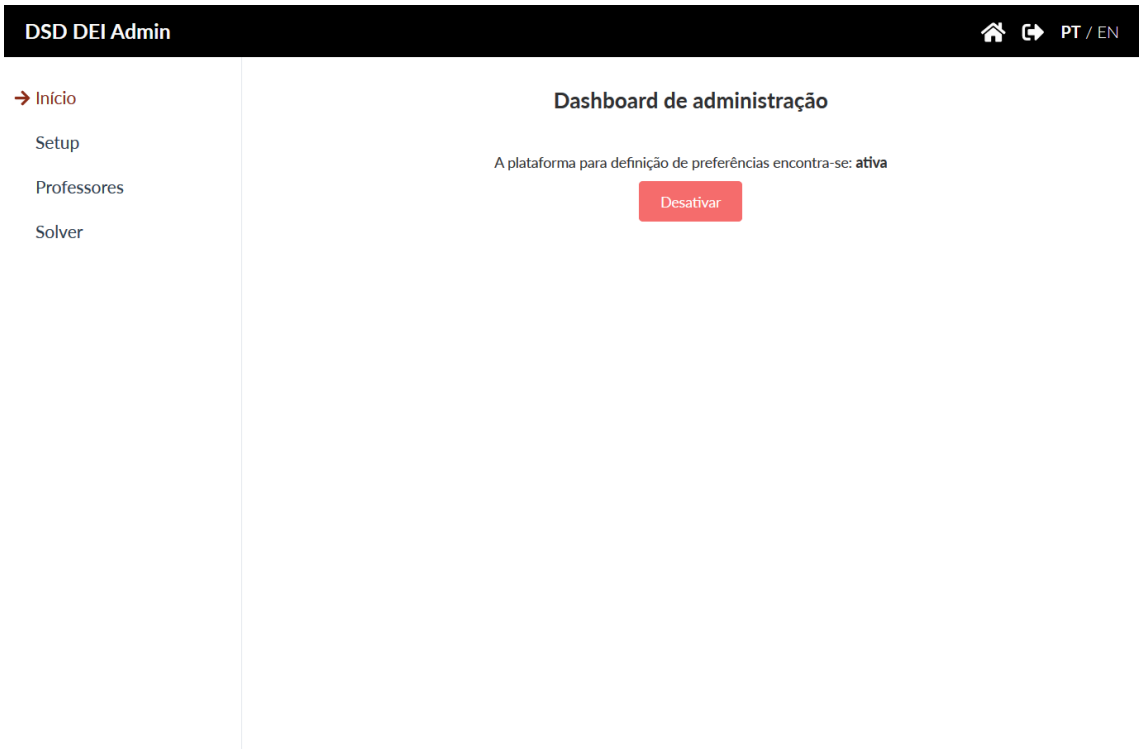

Figura A.14: Ecrã inicial da interface de administração

Na página de *Setup*, ilustrada na Fig. A.15, é possível iniciar um novo ano letivo através da introdução dos ficheiros necessários: professores, atribuições passadas, unidades curriculares, cargos e interrupções planeadas dos professores. Deve-se introduzir o ano a iniciar, que por defeito é o ano atual. Caso se pretenda substituir um ano já existente, devido a um erro na introdução dos dados ocorrido anteriormente por exemplo, deve-se assinalar a opção de "Substituir ano existente". Ao clicar em "Upload"o utilizador deve receber uma notificação de sucesso. Em caso de erro, deverá aparecer o erro ocorrido. A informação relativa ao conteúdo que os ficheiros devem apresentar encontra-se nas Tabelas A.1, A.2, A.3, A.4 e A.5. São também apresentados partes de ficheiro como exemplo para estas mesmas tabelas nas Figs. A.16, A.17, A.18, A.19 e A.20.

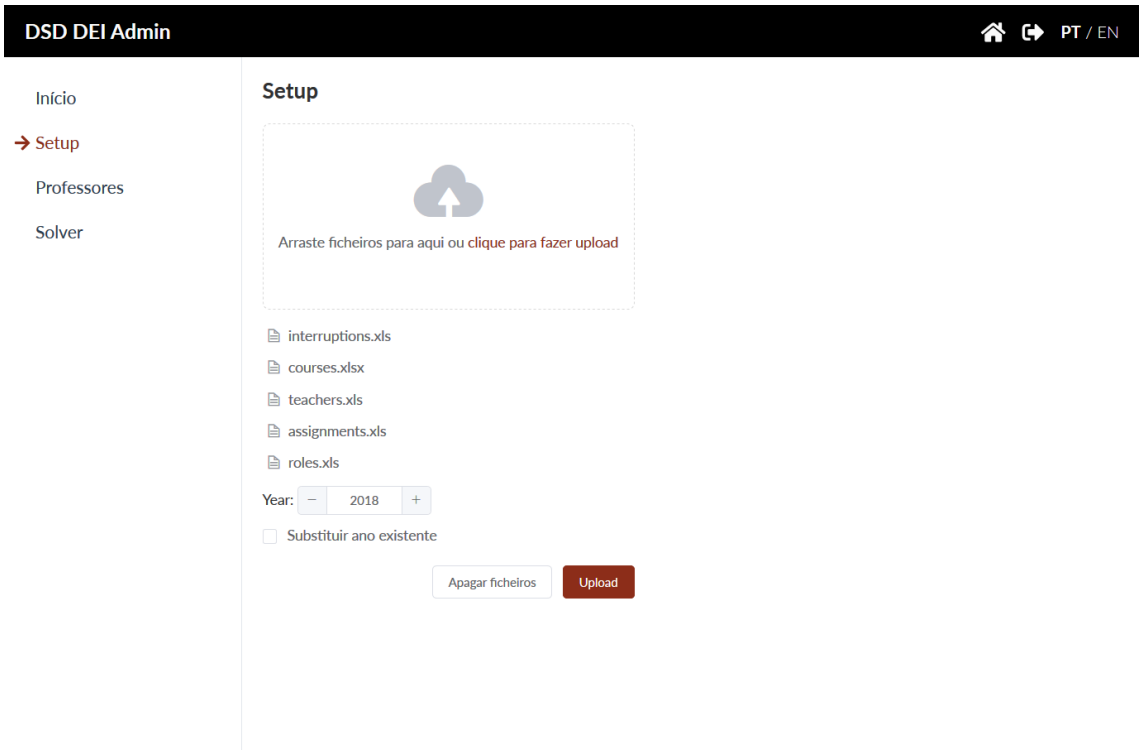

Figura A.15: Página para configuração de ano letivo

Coluna	Significado
Codigo	Código (Número mecanográfico)
Nome	Nome completo
Sigla	Sigla
Unidade	Unidade do professor (ex: Departamento de Engenharia Informática)
Categoria	Categoria (ex: Professor Associado)
%	Percentagem de associação (valor de 0 a 100)
Outra UO	Outra Unidade Orgânica se existente (ex: FLUP)

Tabela A.1: Informação relativa às colunas do ficheiro de professores

	A	B	C	D	E	F	G
1	Codigo	Nome	Sigla	Unidade	Categoria	%	Outra UO
2	201368	Manuel Joaquim	MJBM	Departamento de Engenharia Física	Professor Auxíliar	100	FCUP
3	202873	Augusto da Silve	ASR	Departamento de Engenharia Física	Professor Auxíliar	100	FCUP
4	206415	Raul Fernando d	RMV	Departamento de Engenharia Informáti	Docente aposentado	100	
5	207140	João António Pe	JCB	Departamento de Engenharia Química	Professor Auxíliar	100	
6	207971	Eugénio da Cost	ECO	Departamento de Engenharia Informáti	Professor Catedrático	100	
7	208228	José Augusto Tr	JTB	Departamento de Engenharia Mecânic	Professor Auxíliar	100	
8	208257	Luisa Maria Pim	LMS	Departamento de Engenharia Mecânic	Professor Auxíliar	100	
9	208273	Maria Dulce Soa	DSL	Departamento de Engenharia e Gestão	Professor Auxíliar	100	
10	208327	Armando Luís F	AFL	Departamento de Engenharia e Gestão	Professor Auxíliar	100	

Figura A.16: Exemplo de parte de ficheiro de professores

Coluna	Significado
Area	Área científica (ex: Arquiteturas e Sistemas de Computação)
Curso	Sigla do Curso (ex: MIEIC)
Ano	Ano curricular (ex: 1º)
Semestre	Semestre (ex: 2S)
Código	Código (ex: EIC0020)
Sigla	Sigla (ex: LCOM)
Unid. Curric.	Nome da Unidade Curricular (ex: Laboratório de Computadores)

Tabela A.2: Informação relativa às colunas do ficheiro de unidades curriculares

	A	B	C	D	E	F	G
1	Area	Curso	Ano	Semestre	Código	Sigla	Unid. Curric.
2	Arquiteturas e Sistemas de Computação	LCI	1º	2S	CINF029	SCCOM	Sistemas Computacionais e de Comunicação
3	Arquiteturas e Sistemas de Computação	MCI	1º	1S	MCI0007	SEGINF	Segurança da Informação
4	Arquiteturas e Sistemas de Computação	MESW	1º	2S	MESW000	SES	Segurança em Engenharia de Software
5	Arquiteturas e Sistemas de Computação	MESW	1º	2S	MESW001	CM	Computação móvel
6	Arquiteturas e Sistemas de Computação	MESW	2º	1S	MESW001	COSN	Computação Orientada a Serviços e Nuvem

Figura A.17: Exemplo de parte de ficheiro de unidades curriculares

Coluna	**Significado**
FUNC_CODIGO	Código do professor
ANO_LETIVO	Ano letivo (ex: 2017)
UC_CODIGO	Código da Unidade curricular
TIPO_AULA	Tipo de aula (ex: TP)
FACTOR	Factor da atribuição, entre 0 e 1
ANO	Ano curricular (ex: 4)
SEMESTRE	Semestre (ex: 2S)
HORAS	Horas atribuídas

Tabela A.3: Informação relativa às colunas do ficheiro de atribuições anteriores

	A	B	C	D	E	F	G	H	I
1	FUNC_CODIGO	ANO_LECTIVO	UC_NOME	UC_CODIGO	TIPO_AULA	FACTOR	ANO	SEMESTRE	HORAS
2	207971	2017	Metodologias de Planeamento e Escalonamento	PRODEI017	TP	1	1	2S	2
3	207971	2017	Sistemas Multiagente	PRODEI012	TP	0	1	1S	2
4	207971	2017	Tópicos Avançados em Engenharia Informática	PRODEI042	OT	0	1	1S	1
5	207971	2017	Inteligência Artificial	EIC0029	T	1	3	2S	3
6	207971	2017	Seminário de Sistemas Inteligentes, Interacção e	EIC0101	TP	1	5	1S	2
7	207971	2017	Tópicos Avançados em Engenharia Informática	PRODEI042	OT	0	1	2S	1
8	207971	2017	Planeamento de Investigação	PRODEI036	TP	1	1	2S	0.5
9	207971	2017	Metodologias de Investigação Científica	PRODEI032	TP	1	1	1S	1.5

Figura A.18: Exemplo de parte de ficheiro de atribuições anteriores

Coluna	Significado
LOGIN	Código do professor
CARGO_NOME	Nome do cargo (ex: Diretor Adjunto)
CARGO_CHAVE	Unidade onde se enquadra o cargo (ex: Mestrado em Multimédia)

Tabela A.4: Informação relativa às colunas do ficheiro de cargos

	A	B	C
1	**LOGIN**	**CARGO_NOME**	**CARGO_CHAVE**
2	206415	Avaliador da Comissão Europeia	
3	207971	Coordenador de unidade do Centro Interdisciplinar	Laboratório para a Inovação em Media da U.Porto
4	207971	Coordenador Pedagógico do 1º ano do Curso	Mestrado Integrado em Engenharia Electrotécnica
5	207971	Coordenador Pedagógico do 1º ano do Curso	Mestrado Integrado em Engenharia Química
6	207971	Coordenador Pedagógico do 1º ano do Curso	Mestrado Integrado em Engenharia Informática e
7	207971	Coordenador Pedagógico do 1º ano do Curso	Mestrado Integrado em Engenharia Mecânica
8	207971	Direcção intermédia de 1.º grau	Serviços de Imagem, Comunicação e Cooperação
9	208741	Diretor Adjunto	Mestrado Integrado em Engenharia Informática e

Figura A.19: Exemplo de parte de ficheiro de cargos

Coluna	Significado
N_MEC	Número mecanográfico do professor
INTERRUP_NOME	Nome da interrupção (ex: Licença sabática)
D_EFEITOS_INTERRUP	Data de início (ex: 16/08/2017)
D_FIM_INTERRUP	Data de fim (ex: 15/02/2018)

Tabela A.5: Informação relativa às colunas do ficheiro de interrupções

	A	B	C	D
1	**N_MEC**	**INTERRUP_NOME**	**D_EFEITOS_INTERRUP**	**D_FIM_INTERRUP**
2	419241	Licença sabática	16/08/2017	15/02/2018
3	209566	Licença sabática	16/02/2018	15/08/2018
4	234149	CT Comissão de serviço	21/11/2011	
5	310021	Licença sabática	16/08/2017	15/02/2018
6	246626	Licença sabática	16/02/2018	15/08/2018
7	237766	Licença sabática	16/02/2018	15/08/2018

Figura A.20: Exemplo de parte de ficheiro de interrupções

Na página para configuração de professores, presente na Fig. A.21, e possível retirar professores que lecionaram no ano corrente mas que não lecionarão no seguinte. A *checkbox* deve ser desselecionada nos professores a remover. Para gravar as alterações é necessário clicar no botão "Gravar"no final da página.

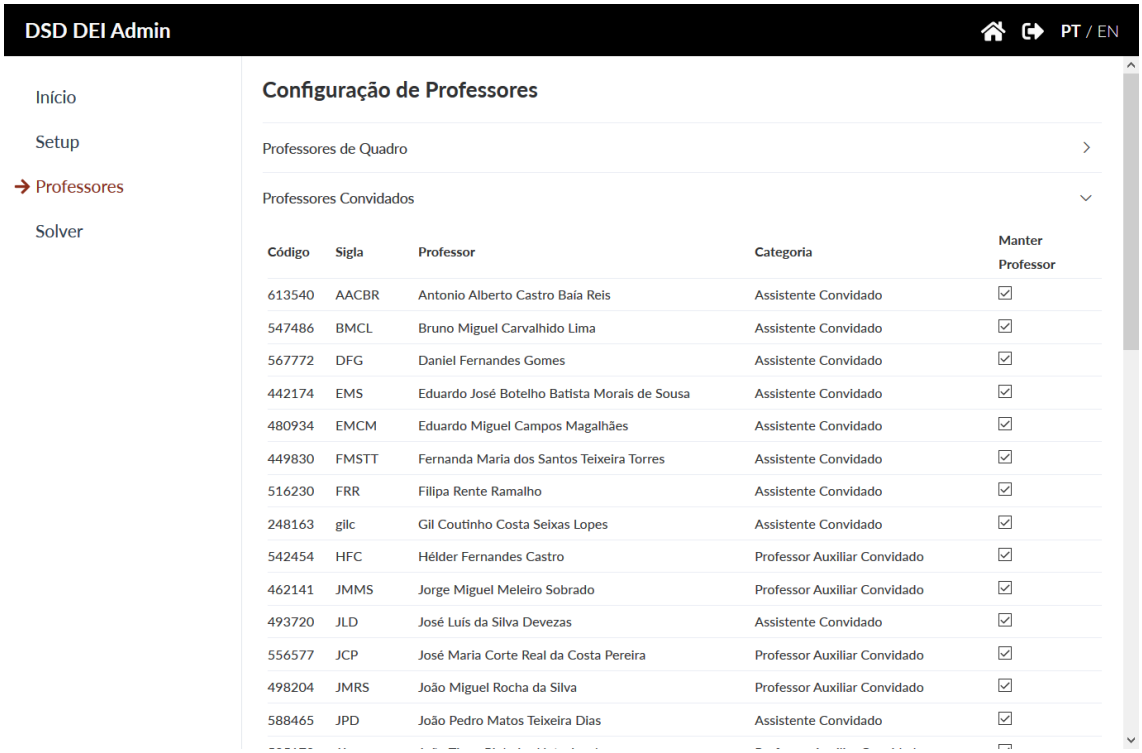

Figura A.21: Página para configuração de professores

Já na página do Solver, ilustrada na Fig. A.22, é possível fazer upload do ficheiro de pedido de serviço docente. Ao fazer upload, deverá aparecer mensagem de sucesso no caso do carregamento do ficheiro ser bem sucedido. Em caso de erro, aparece uma mensagem com o motivo do mesmo. A informação relativa ao ficheiro de entrada e colunas necessárias encontra-se na Tabela A.6, com um exemplo na Fig. A.23.

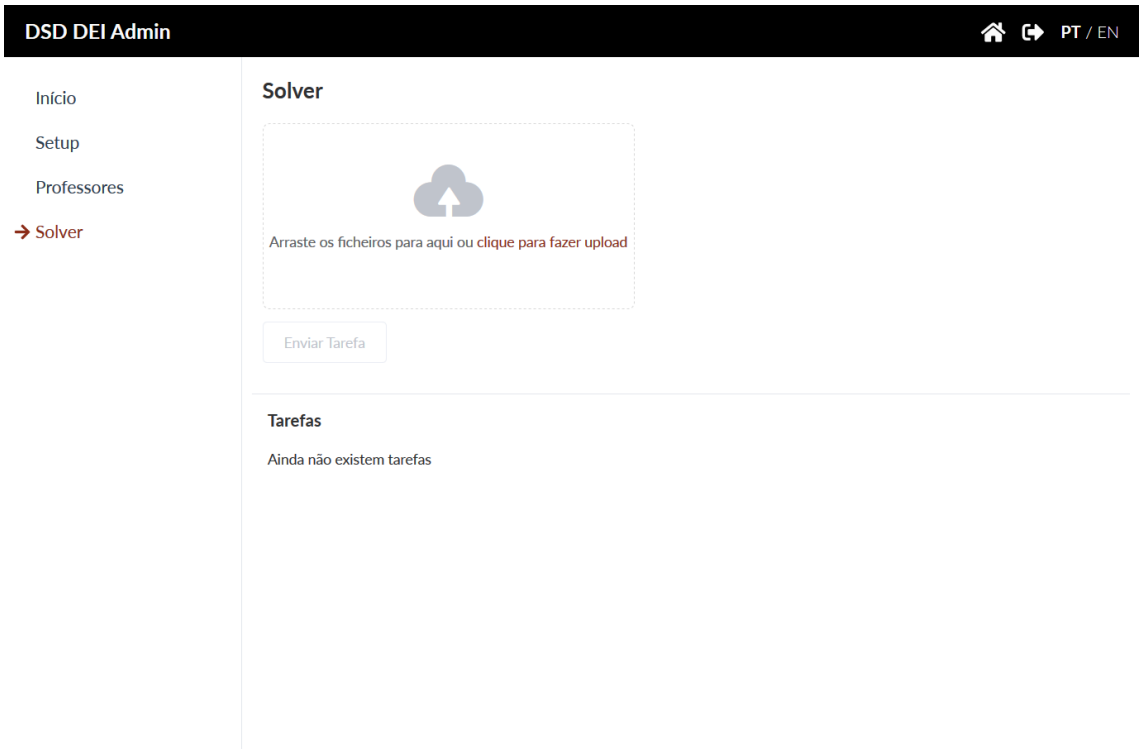

Figura A.22: Página do solver

Coluna	Significado
Semestre	Semestre da unidade curricular (ex: 1)
Código	Código da unidade curricular
Tipo	Tipo de aula (ex: TP)
HorasTurma	Horas por turma (ex: 2)
Turmas	Número de turma requisitado ao departamento (ex: 5)
Doc	Docentes pré-atribuídos, com sigla ou código e número de horas opcional, separados por ";" (ex: AFCC (1h); RPR (1h))
Factor	Fator a atribuir, entre 0 e 1
Profs. Externos	Professores externos a pré-atribuir, com formato semelhante à coluna "Doc"
Convidados	Convidados a pré-atribuir, com formato semelhante à coluna "Doc"

Tabela A.6: Informação relativa às colunas do ficheiro de pedido de serviço docente

	A	B	C	D	E	F	G	H	I
1	Semestre	Código	Tipo	HorasTurma	Turmas	Doc	Factor	Profs. Externos	Convidados
2	1	MIET0023	TP	3	0.166666667		1		
3	1	MM0027	TP	3	1		1		RLNP
4	1	MM0038	TP	3	1		1	HMPPCA	
5	1	MM0040	TP	3	1		1	CSLM (1h)	LM (2h)
6	1	MM0043	TP	3	1		1	JMSFCC	
7	1	MM0047	TP	3	1	AOR	1		
8	1	MM0054	L	3	1		1		CMMOPS
9	1	MM0055	TP	3	1		1	JCMP(1h)	EMS (2h)
10	1	MM0057	TP	3	1		1		SAMF
11	1	MM0060	TP	3	1		1	PFC (1h)	SAMF(2h)
12	1	MM0061	TP	3	1		1		RPSN
13	1	MM0064	TP	3	1	AFCC (1h); RPR (1h)	1		

Figura A.23: Exemplo de parte de ficheiro de pedido de serviço docente

Após upload bem sucedido, deverá aparecer a tarefa a realizar, com o estado pendente, como é visível na Fig. A.24.

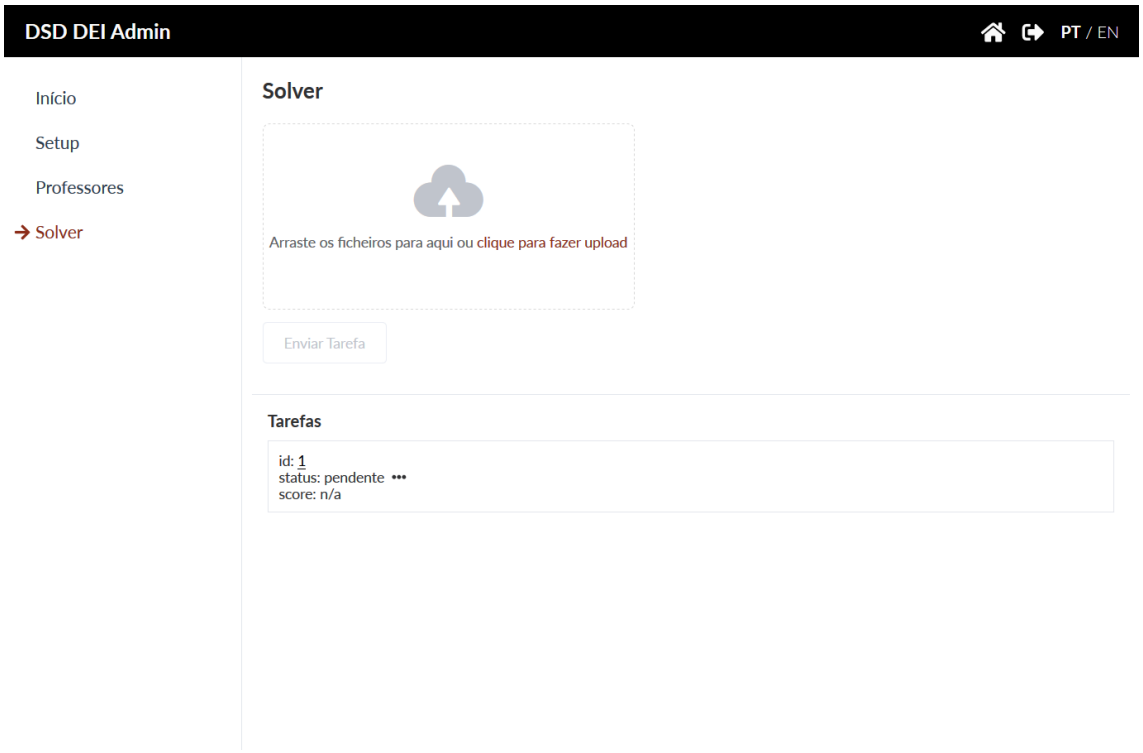

Figura A.24: Início de nova tarefa do solver

Logo de seguida, o Solver deve iniciar a resolução do problema e a tarefa muda automaticamente para o estado "a correr", como é possível observar na Fig. A.25. O valor da função de avaliação também vai sendo atualizado em tempo real.

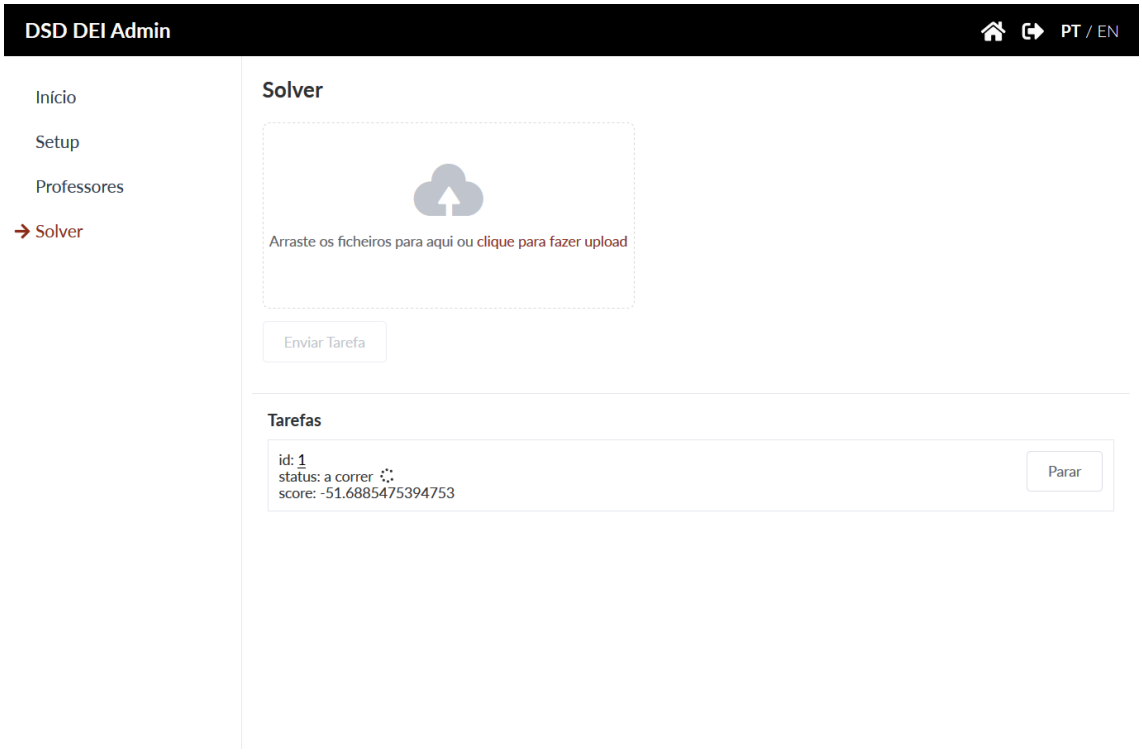

Figura A.25: Tarefa iniciada

Quando o valor obtido for satisfatório, a tarefa deve ser parada através do botão "Parar". A tarefa muda de estado para "a parar", como é possível observar na Fig. A.26.

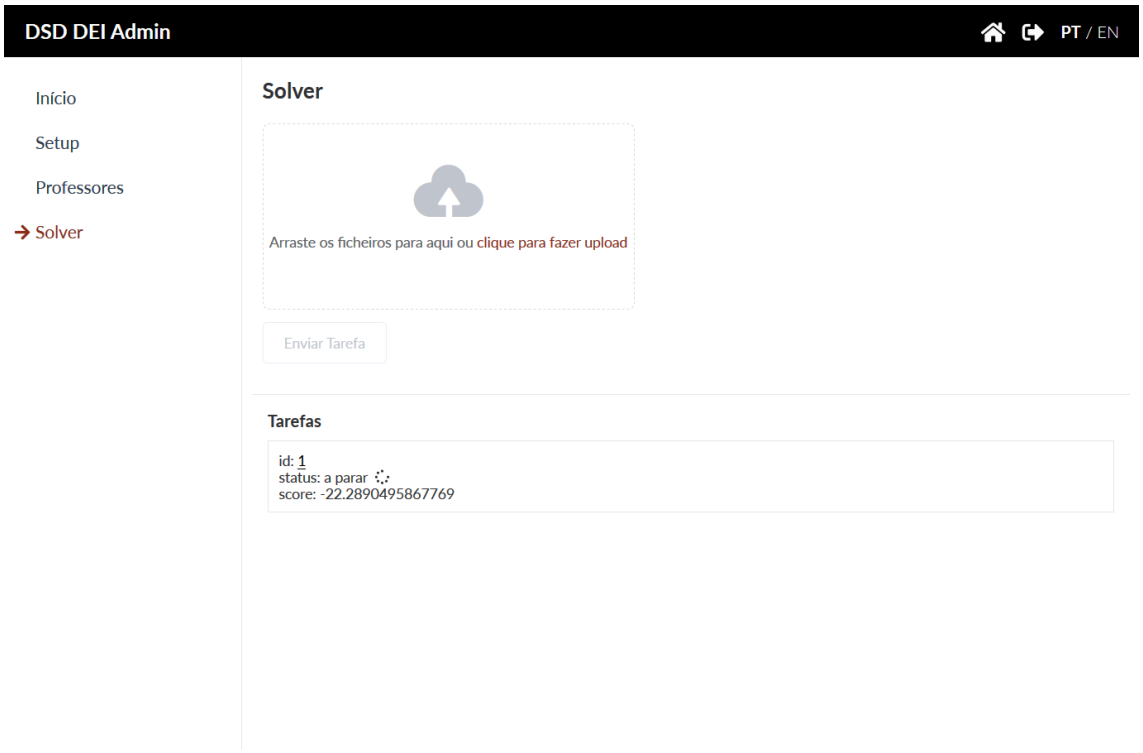

Figura A.26: Tarefa a terminar

Passado alguns segundos, a tarefa deverá mudar para o estado de sucesso, como ilustrado na Fig. A.27. Surge também um novo botão que permite exportar os dados para um ficheiro do tipo CSV, para importação no SIGARRA.

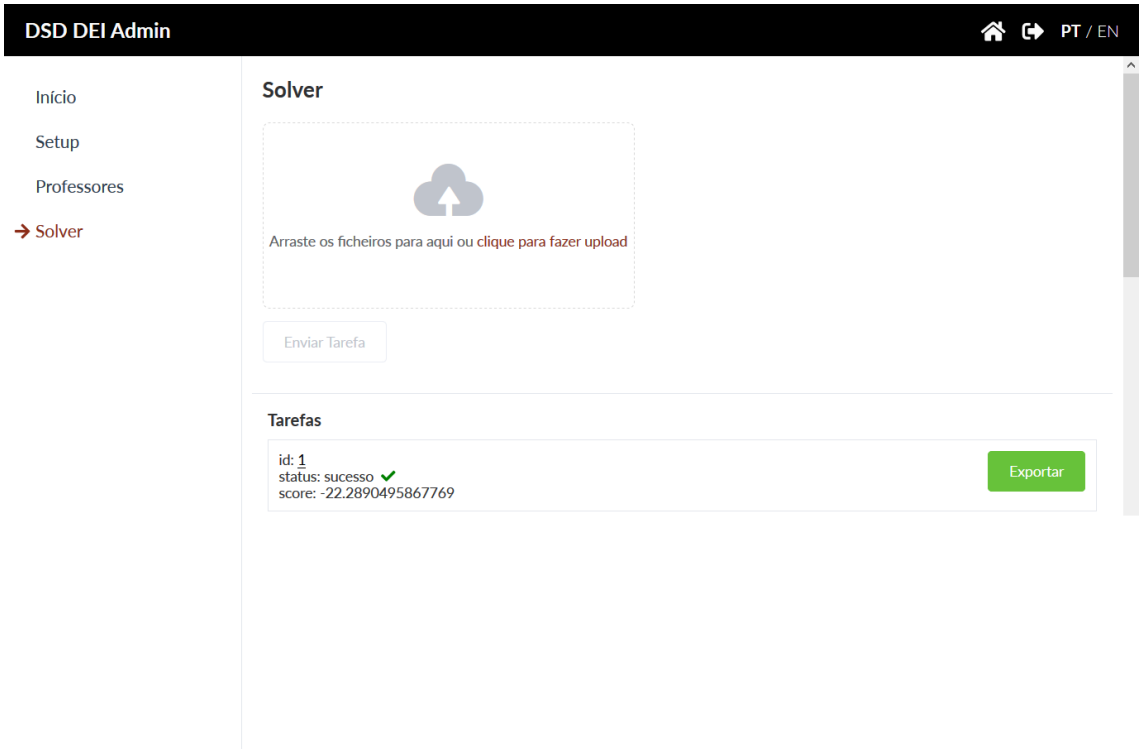

Figura A.27: Tarefa bem-sucedida

Ao clicar numa das tarefas bem-sucedidas, é possível editar a mesma através da interface que pode ser observada na Fig. A.28. Através desta matriz, e através de duplo-clique numa das células, pode-se alterar o valor tanto das horas como do fator respetivo. Para além de editar valores já existentes, que podem ser colocados a zero para remover uma atribuição, é possível também fazer novas atribuições.

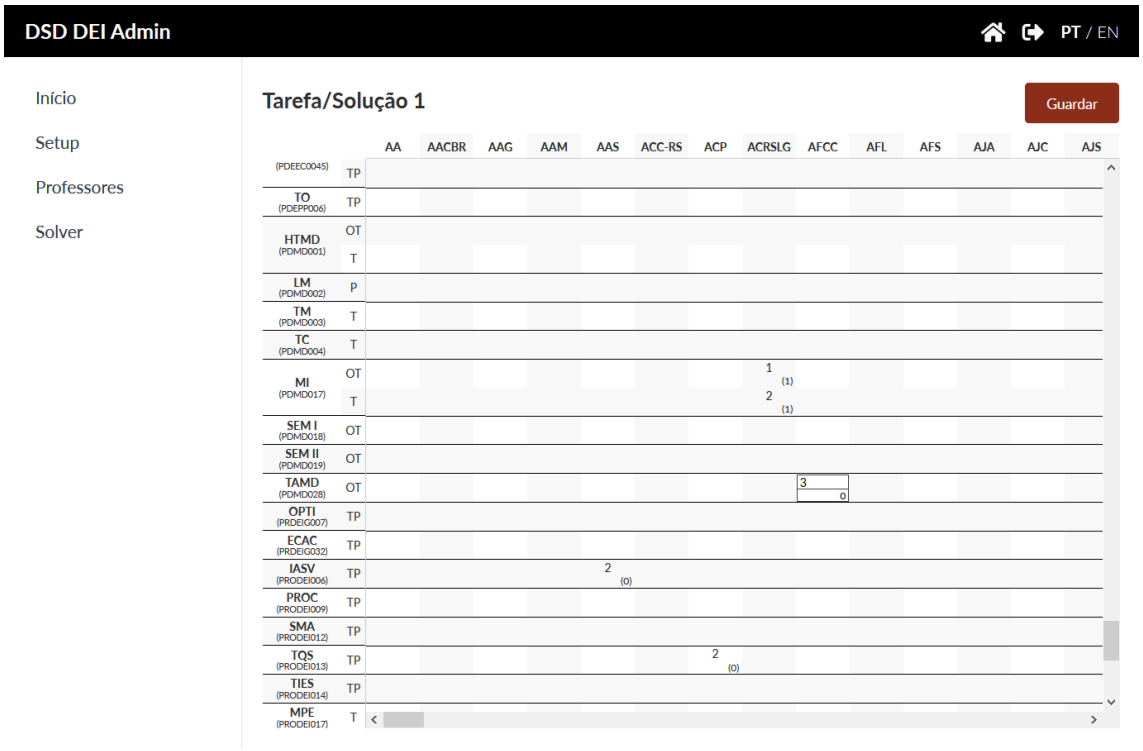

Figura A.28: Interface para edição de atribuições, com alteração em progresso

Ao editar um valor de horas, surge um *pop-up* que permite a transferência de horas. No caso de um número de horas ter sido reduzido, é possível escolher outro professor a quem atribuir essas mesmas horas. No caso de aumento, como demonstrado na Fig. A.29, o sistema pergunta se pretende reduzir esse número de horas a outro professor presente na unidade curricular.

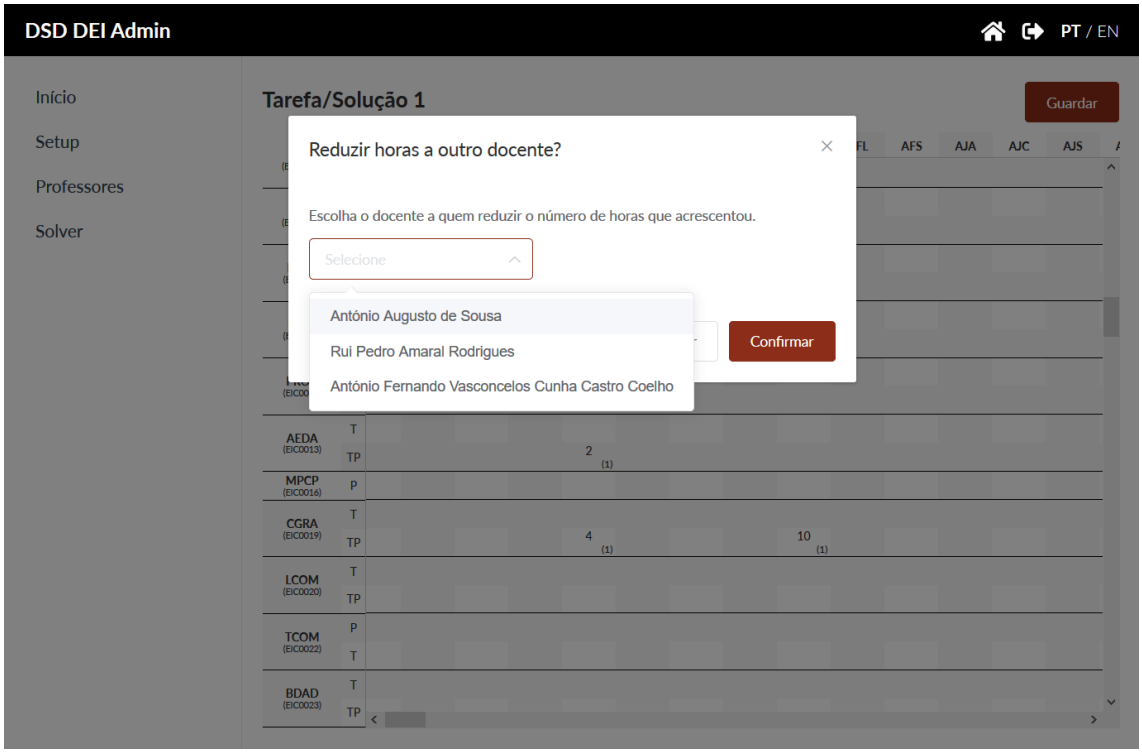

Figura A.29: *Pop-up* para auxílio de transferência de horas